RealParse 2.7

Table of Contents

Overview 2
Installing RealParse 2
  System Requirements 2
  Installation 3
    Core Installation 3
    Configuration Installation 3
XML Configuration 4
    RealConfig.xml Configuration 4
    Layout.xml Configuration 6
  Optional Configuration 11
    Filters_Custom.pl Configuration 11
    RealFeatures.pl Configuration 12
  Other Installation Requirements 13
    Data Description Values 13
Administrator Maintenance 15
  Configuration Files 15
  Error Logs 15
  Recommended Cron Setup 15
  External Data Dependencies 15
  Program Recovery Procedures 15
Troubleshooting Guide 16
Support Information 16

Overview

RealParse is a filtering tool for real estate data feeds. It converts the data into a format that the Real Estate Loader can understand. This manual will cover the RealParse tool for use with real estate data.

RealParse uses configuration files, written in XML format, to understand delimited data files containing real estate data. It then follows the instructions of the XML file to parse the real estate data into the correct Real Estate Loader format for loading into the database. RealParse has a set of text filters that it can apply to the raw data to filter almost any type of text data. Newer custom filters can be made to extend the ability of the parser to handle virtually any format of data.

Installing RealParse

Supported Platforms

The only requirement for the platform is that it must be able to run Perl. The *recommended* platform would depend on the amount of parsing which the RealParse tool will be required to perform. A Unix/Linux platform is best for most situations. Macintosh OSX has not been tested, but since it does support Perl, it should be acceptable.

System Requirements

The type of system (hardware) you would want will depend on the amount of data that is to be parsed. The things to think about when selecting the system is the CPU speed, IO abilities, and Storage. The CPU is heavily used while RealParse is running. After all text has been parsed, image files are then copied to a temporary location for later use. This requires a decent amount of IO cycles if there are a lot of images with the data. Storage is probably the most important. There will need to be enough space to hold the raw data files, image files, and then the copied image files. This all depends on the data being parsed, but usually this gets up to around a few hundred MB if images are sent with the feed. To help save space, a utility for searching for unused and old files can be very useful, and you can usually obtain one of these utilities from your system administrator, or have your sys admin install the utility himself. An age date of about 15 days is usually a good setting for this utility. Once the files have reached an age of 15 days, they will be removed by that utility to help save space.

Installation

There are two types of installations, the main installation of the RealParse core, and the configuration installation. The RealParse core only needs to be performed once on each physical machine, while the configuration installation needs to be performed once for each new set of real estate data that needs to be parsed on that server. The files that come with this release of RealParse are:

- **RealParse-core.tar.gz:** The files that make up the core install. These are required.
- **RealParse-conf.tar.gz:** The configuration files. These files are example files and should only be used as a base for creating new property installations.
- **RealParse-conf_optional.tar.gz:** More configuration files. These files are not required, but could be customized and used for custom features.
- **RealParse-examples.tar.gz:** Some older example files to use as a reference.

Core Installation

- **Step 1:** Determine the location of where RealParse should be. This is usually best if it's on the same machine as the real estate data. At the very least, it must have read access to the real estate data. For this document, we'll assume the location is, "/bin/realparse/".
- **Step 2:** Determine the location of where the RealParse config files should be stored. This should be in a location that is backed up to make sure these are not lost. For this document, we'll assume the location is, "/var/realparse/".
- **Step 3:** Extract the RealParse core files (*these files are never modified*) to the "/bin/realparse/" directory. These files should be provided as a tar.gz file named, *RealParse-core.tar.gz*. As of this writing, these files are:
  - RealParse.pl – This file is RealParse. You pass it variables informing it what to parse and how to parse it.
  - genericFileSearch.pl – This file searches for the raw data files in the directory specified in the RealConfig.xml file. Look up the <*Search Include*> element under the RealConfig.xml Configuration section.
  - Filters_Base.pl. - This is the base set of filters for RealParse.
- **Step 4:** Extract the sample configuration files to the "/var/realparse/" directory. These files should be provided as a tar.gz file named, *RealParse-conf.tar.gz*. This archive contains:
  - RealConfig.xml – Contains most of the property configuration variables to allow RealParse to run. This is the file that is passed to RealParse during execution.
  - layout.xml – Layout file which describes how to parse the raw data.
- **Step 5:** Extract the other sample configuration files, if your properties will be using them, to the "/var/realparse/" directory as well. These files should be provided as two tar.gz or a zip files named, *RealParse-conf_optional.tar.gz* and *RealParse-examples.tar.gz*. These files are not required, but some property configurations may find them useful.

Configuration Installation

- **Step 1:** Create a new directory under the "/var/realparse/" directory. (*This directory is just for example use in this document, substitute your configuration directory in its place*) For example, we'll assume the property's name is "Bob's Big Houses". The directory would then be something similar to, "/var/realparse/bbh/".
- **Step 2:** Copy the two XML config sample files into the "/var/realparse/bbh" directory. These sample files should be located in the, "/var/realparse/", directory: layout.xml and RealConfig.xml. Rename these files to something meaningful for the type of real estate data this installation will be parsing, like *layout_residential.xml* and *RealConfig_residential.xml*. Do not worry about copying any other sample files at this time, they are only used if you enable certain options in the RealConfig.xml file.
- **Step 3:** Configure the two XML files for this new real estate feed. More detail on the configuration should be contained in this document.

XML Configuration

There are two main XML configuration files that you must edit for each installation of the RealParse. Both of these files are stored in XML format to allow for easy editing and parsing. These files should already be in your configuration directory, and all that should be required is customizing them for the new real estate feed.

RealConfig.xml Configuration

This file's name usually begins with, *RealConfig,* and is a very simple XML structure that contains comments to help with configuring. Here is a detailed description of the elements:

- <RealParse>: This element is the root element and contains two attributes; showErrors and showDebug. These two attributes are toggled on or off with either a "0" (off) or a "1" (on).
- <Includes>: This element is always within the <*RealParse*> element and does not contain any attributes. This element is used only for organization, and will contain the elements that deal with external files.
  - <Pre_Process>: This will contain <*Command*> elements that contain a single line of text that will be executed by the parser before it actually does any parsing. For this reason, you should make sure you have tight permissions on this configuration file so that unauthorized people can't setup commands to be executed by RealParse. Also make sure permissions are set tight enough so that only the people you want to read this file are the ones you would be comfortable with knowing passwords, because sometimes you have to place login information into a command. You may want to ask your system administrator for suggested permissions.
  - <Post_Process>: This will contain <*Command*> elements that contain a single line of text that will be executed by the parser after all parsing is complete. For this reason, you should make sure you have tight permissions on this configuration file so that unauthorized people can't setup commands to be executed by RealParse. Also make sure permissions are set tight enough so that only the people you want to read this file are the ones you would be comfortable with knowing passwords, because sometimes you have to place login information into a command. You may want to ask your system administrator for suggested permissions.
  - <Code_Include>: This is an included Perl file that contains code for searching feature sets. This is not required, but is very useful when you have a Feature column in your data feed. There are two example versions that are provided: RealFeatures_1d.pl and RealFeatures_2d.pl. The 1d file handles one dimension feature sets, while the 2d file handles two dimensional feature sets. These two example feature parsers will probably not work for your current installation and you will need to write a custom version and place it in the properties configuration directory. (Please look at the **Optional Configuration** section for the **RealFeatures.pl Configuration** information.) This field requires a full path.
  - <Layout_XML>: This is the location of your other XML configuration file. We will discuss this in greater detail later on. This requires a full path.
  - <Search_Include>: This is an included Perl file that searches for files that are specified in the <*Input_Filename*> field. There is a standard file search script provided with the core installation: genericFileSearch.pl. This can always be replaced with a custom search routine, but when using a custom file, keep the file within the property's configuration directory. This field requires a full path.
  - <Filter_Include>: This will have <*Filter*> elements that contain the full path to filter files. There is a filter file that comes with the Core Installation named, Filters_Base.pl. You can add more <*Filter*> elements to specify more filter files. The format for these files will be discussed in greater detail in the **Optional Configuration** later in this documentation.
  - <Pre_Output_Filename>: This element contains a full path to a file whose data will be placed at the beginning of the output file specified with the <*Output_Filename*> element. There's also one attribute with this element, *delete*, which if set to "true" will delete the file after it has been used. If "false" then it will not be removed.
  - <Post_Output_Filename>: This element contains a full path to a file whose data will be placed at the end of the output file specified with the <*Output_Filename*> element. There's also one attribute with this element, *delete,* which if set to "true" will delete the file after it has been used.

If “false” then it will not be removed.

- <Config>: This element is always within the <*RealParse*> element and does not contain any attributes. This element is used only for organization, and will contain the elements that deal with internal configuration of RealParse.
  - <Loader_Version>: This was created because more than one version of the Real Estate Loader was being used with the same parser. Most loaders will be version 2.0+ but in case they are not at least version 2.0 then you need to set this to a lower number. Changing this number will change the format of the output that the parser will write. The accepted version numbers, and what they do, are:
    - Loader Version 1.0 will output to the old format that is rarely used.
    - Loader Version 1.1 changes bedrooms to beds and bathrooms to baths.
    - Loader Version 1.15 changes garage from a simple feature to a feature and quantity.
    - Loader Version 1.16 moves carport from the garage section to its own section with feature and quantity.
    - Loader Version 2.0 outputs extended agent/broker info as well as all the previous version changes.
    - Loader Version 2.1 does away with the Business tags such as “Business ID” or “Business Name” and outputs “Broker ID” and “Broker Business Name” in their place. For new installations, use this 2.1 value.
- <Lock_File>: This file is created when RealParse begins, and then deleted when RealParse completes successfully.
- <Data_Dir>: This is the directory location of the raw, real estate data. This should be a full path.
- <Image_Dir>: This is where the images will be copied to when the parser is running. This location will later be used by the loader, to load images into the database or to an image server.
- <Image_Handling>: This field instructs RealParse to either move or copy the images from the raw data location to the Image Directory. “Move” and “Copy” are the only valid options for this field. Copy is useful for testing so that you don’t have to reload your images each time, and Move is useful for live setups to save hard drive space.
- <Image_Format>: Most image filenames are an abbreviation of the realtor, followed by the MLS number, followed by a number or letter to increment images that are for the same house. All of that is then followed by the standard “.jpg” or “.gif” file extension. This field lets you specify how the image filenames are laid out. There are some strings that will be replaced in this field as it looks for the image files, these are:
  - #mls# - This string will represent the location of the MLS number in the filename.
  - #letter# - This string represents the incremental letter in case of multiple images.
  - #number# - This works similar to #letter# for images that use number increments instead.
  - #extra# - This represents the location of the extended image labels. Thumbnails are a good example of a use for this.
  - Attributes:
    - fullsearch – Value must be either “true” or “false”. If true then it will search for all possible images. If false then it stops looking for incremental images after it fails to find two. So if you plan on having images such as, image1.jpg, and, image8.jpg, with 2 through 7 not included, then you should enable a full search. If in doubt, just set this value to true.
    - mls – This usually just contains the text, “#mls#”.
    - letter – This usually just contains the text, “#letter#”.
    - number - This usually just contains the text, “#number#”.
    - extra - This usually just contains the text, “#extra#”. This attribute is used for extended image searching. To read more about this, read the next element, *Extra_Images*.
- <Extra_Images>: This element is used only with the new <*Image_Format*> element. The attribute, *extra*, will be replaced with the value from the attribute, *fileText*, from each of the internal <*Include*> elements. The *categoryText* attribute is the text that is used when RealParse generates the output that is loaded in the database.
- <Input_Filename>: This field should not be a full path. It is the filename of the raw data file that will be searched for. There are two attributes inside this element that can be set: delimiter and backup. The delimiter attribute is usually a tab (\t) or a pipe (|), but can be any character. The backup attribute tells RealParse to backup the raw data files for easy error checking. The only values allowed in the backup attribute are 0 and 1, with 1 meaning true.

- <Output_Filename>: This is the output filename where the parsed output is written. This file is rewritten each time RealParse runs for this property. This should be a full path.
- <Output_List_Filename>: This file will be a list of all the files output by the parser. These files include the <*Output_Filename*> and the images for it. The parser writes this at the very end of the parsing process, so it should be possible to use this as a way to determine if the parser has run and if it is finished. If the file does not exist then do not run the Real Estate Loader and check back later, and if it does, then it should be safe to run the Loader. This needs to be a full path as well but this is not required and can be safely commented out if you do not wish to have this file created. This element contains five attributes:
  - *prefix* and *postfix*: While creating the list file at the end of the parsing process, the text within the prefix field will be written to the beginning of each filename and the text in the postfix will be written to the end of the filename. This is useful if you want to use the list file as a sort of script for moving files around or for doing further processing.
  - *owner* and *group*: These set the owner (chown) and group (chgrp) of the list file to allow for other processes running under other permissions to access it.
  - *mask*: This is the octal number used for setting the permissions (chmod).
- <Log_Filename>: The full path of where the log file should be written. There is one internal value called, append, and if set to "1" or "true" or "yes" then it will append to the log file. If not set to one of these true values, then it will overwrite the log file each time the parser runs. There are also two other options that work with the log file, and they are located at the top of the RealConfig file where the first element, <*RealParse*>, is located. You'll see that it has two attributes:
  - *showErrors*: Set this to 1 if you want errors printed to the screen or log file, or 0 if you don't want to see errors.
  - *showDebug*: Set this to 1 if you want debug messages printed to the screen or log file, or 0 if you don't want to see debug information.
- <Property>: This is the element that is predefined by the real estate setup on the database side. Make sure this field is correct or the data will not be loaded under the correct property.
- <Category>: This field is normally not required but in some special cases, it may need to be changed, so it is provided for this reason. If you are unsure about this, just completely remove it, comment it out, or set its value to, "REAL ESTATE".
- <Ad_Type>: This should be one of six types of real estate data, depending on the raw data that is being parsed. The options are: RESIDENTIAL-LISTING, RESIDENTIAL-RENTALS-LISTING, COMMERCIAL-LISTING, COMMERCIAL-RENTALS-LISTING, LAND-LISTING, MULTIFAMILY-LISTING, MULTIFAMILY-RENTALS-LISTING, APARTMENT-LISTING, AUCTION-LISTING, NEWHOME-BHI-LISTING, NEWHOME-LISTING, VACATION-LISTING, VACATION-RENTALS-LISTING, FARM-LISTING, FARM-RENTALS-LISTING.
- <Custom>: This element is always within the <*RealParse*> element and does not contain any attributes. This element is used only for organization, and will contain the elements that deal with custom configuration of tools used with RealParse.
  - <Reference_List>: This is a Perl script that contains a large, possibly multi-dimensional, array or hash, which is a feature set. The feature set is a large list of features in our database such as, cable TV, pool, washing machine, and other similar amenities. Editing this file for the feature set of the raw data is the most time consuming process. You must match up each feature value with a valid value from the real estate database. This field requires a full path.

Layout.xml Configuration

The layout.xml file is fairly simple in its design but allows a high level of configuration for how the raw data should be parsed. The file is created and modified using a tool called, Match, which allows easy drag-n-drop customization of the layout file. At the time of this writing, this tool is not stable enough for release, so any modifications needed to these files will need to be performed by hand at this time until the Match tool is released.

Here are some guidelines for understanding and editing the layout.xml file:

- The first xml element (root element) should always be: <Match>
  - This element can also have a version variable inside of it to specify the version of the Match utility that created the file. If a version is not specified, then the parser will assume it was hand made without the use of the Match utility. *(At this time, RealParse does not function differently depending on versions, but this may change in the future.)*

- Below the <Match> element there can be multiple <Config> elements that configure certain filters. The <Config> elements are only there if they are required by a filter.
  - When working with the <Config> blocks, keep in mind that these are like global variables. A single Config block can only be linked to one type of Filter, but it will apply to all of the filters of that type.
  - Here is an example of a general <Config> block:

    <Config type="FilterName">

    <!-- filter config data goes here using xml elements specified by the Filter -->

    </Config>
    - The "FilterName" would be replaced by the actual name of the Filter for which this <Config> block was created.

- After the <Config> elements come the <Data> elements. Here is an example:

  <**Data description**="**Agent Name**">

  </**Data**>
  - The value in the quotes must match one of the predefined values for the database. A list of these values is listed under the **Other Installation Requirements** section in this document.

- Inside each <Data> block, there should be a <Source> block. If no filters need to be applied to the raw data that this element represents, then only put the column number of where the data can be found in the raw feed between the Source elements like so:

  <Data description="Agent Name">

  <Source>4</Source>

  </Data>
  - This will take whatever data is in column four of the raw feed and save it to the file that will be imported into the database.

- Filters are specified in the <Source> elements, and <Filter> elements can only contain <Source> elements. Here is a simple example:

  <Data description="Agent Name">

  <Source>

  <Filter type="Case" direction="upper">

  <Source>4</Source>

  </Filter>

  </Source>

  </Data>
  - The type value is the name of the filter and is required by every <Filter> element. Any settings after the type are for the use by the filter only, so they will change from one filter to the next.

- Filters can be placed within other filters like so:

  <Data description="Agent Name">

  <Source>

  <Filter type="Case" direction="upper">

  <Source>

  <Filter type="Case" direction="lower">

  <Source>4</Source>

  </Filter>

  </Source>

  </Filter>

  </Source>

  </Data>
  - The above block of XML will take the data in column four and convert it to lowercase, and

then convert that lower case data into upper case.

- Filter types that are currently allowed are:
  - Addition: Only works with number values, not text. Will add together the Sources into a final value.
  - Append: Will take all of its Sources and append them into one large value and will use a delimiter if you set it.
  - Case: Converts text into upper or lower case.
  - Priority: Will contain multiple <Source> elements that it will check in order for data. If the first does not contain data then it checks the next one. The first <Source> that returns data will be returned by this filter.
  - Reference: Acts much like a hash table. It will use the data as the key, and if the key has been defined then it will return the value that the key represents. This filter requires a <Config> block to function.
  - StaticValue: Ignores any internal <Source> elements and returns a static value independent of the data feed.
  - Strip: *Deprecated* Will strip text from the data. This filter has been replaced by the SearchReplace Filter, and should not be used by new feeds. Old feeds can continue to use this but it will not follow through into version 3.0 of the RealParse package.
  - SearchReplace: Basic search and replace which supports regular expressions.
  - TextToNum: Converts words such as, “five”, to the number, “5”.

- Addition filter structure example:
  <Filter type=”Addition”>
  - This filter adds the Source values within it. It can hold one or more Source blocks and does not have any extra setting values. If it encounters text, it sets the text to 0 and continues adding the rest of the Sources. An example of this would be when you need to add upstairs bedrooms with downstairs bedrooms to get the total number of bedrooms for a house.

- Append filter structure example:
  <Filter type=”Append” delimiter="|">
  - This filter can have one or more Source elements that it will append together. The delimiter is placed between the Source text. Example use would be to append First Name and Last Name with a delimiter of a space, to create the Full Name.

- Case filter structure example:
  <Filter type=”Case” direction=”upper”>
  - This converts text to all upper case or all lower case. The direction value can be either “upper”, or “lower”. Only one Source is allowed within a Case filter. This filter can have more than one Source.

- Priority filter structure example:
  <Filter type=”Priority”>
  - Will contain multiple <Source> elements that it will check in order for data. If the first does not contain data then it checks the next one until it finds a Source with data or runs out of Sources.

- Reference filter structure example:
  <Filter type="Reference" reference="GarageType">
  - This is the only filter at this time to require <Config> values. Along with this filter you must have Config blocks at the beginning of the XML file that describe the references for this filter to apply against the data. Within the Filter element, the reference value is the label of the Config reference that the filter should use with this data. In this example, the reference is GarageType, so at the beginning of the XML file there would be a block that looks like this:

```
<Config type=”Reference”>
        <Label>GarageType</Label>
        <Reference>
                <Key>DE</Key>
```

```
                <Value>Detached</Value>
            </Reference>
            <Reference>
                <Key>AT</Key>
                <Value>Attached</Value>
            </Reference>
        </Config>
```

- This Config block, along with the reference filter, will convert a data value of DE to Detached. Keep in mind that the Config block by itself cannot modify data, it only instructs the filter, which it represents, on how to modify the data. Both the Config and the Filter blocks are needed.

❖ StaticValue filter structure example:

<Filter type=”StaticValue” value=”NA”>

- This filter completely ignores any Source elements within it. It will always return the data in the “value” variable. So in this case, it will always return a string of, “NA”.

❖ Strip filter structure example (DEPRECATED – USE SearchReplace FILTER INSTEAD):

<Filter type=”Strip” allLetters="false" allNumbers="false" spaces="true" newlines="false" custom="">

- This strips character(s) from the data. It has a few true/false variables to make it easy to strip common characters:
  - ❑ allLetters: will strip all non-number characters
  - ❑ allNumbers: will strip all numbers
  - ❑ spaces: will strip all white space
  - ❑ newlines: will remove any new line characters (currently no use for this)
- Along with the true/false variables, you can have more control over which characters are stripped by using a regular expression in the custom variable. For example, you could strip all o's from the data, so that if the data is “monkey”, you are left with “mnkey”. This field will also accept regexp searches such as, “[abc]bob”, which will remove all text that starts with either a, b, or c, and followed by bob.
- Only one Source is allowed within a Strip filter.

❖ SearchReplace filter structure example:

<Filter type=”SearchReplace” regexpSearch=”yes” regexpBackRefs=”no” globalReplace=”no” ignoreCase=”no” search=”abc” replace”123”>

- This filter will search for a string or regular expression and replace it with the given value. Here is a description of the internal variables:
  - ❑ regexpSearch: set to true, yes, or 1 to enable regular expression searching.
  - ❑ regexpBackRefs: set to true value if you would like to use back references in your replace string, such as $1 for the first set of parentheses in the regular expression search.
  - ❑ globalReplace: set to true value if you want to replace every instance or a false value if you want to replace only the first instance.
  - ❑ ignoreCase: set to true value if you want to ignore case while searching.
  - ❑ search: plain text with optional regular expression to search for.
  - ❑ replace: plain text with optional regular expression back references.

❖ TextToNum filter structure example:

<Filter type=”TextToNum”>

- This filter will try to convert text like “one” into the number 1. It will only accept one Source block. Some strings like “1 and a half” will come out as 1.5, but there are some strings that it cannot filter.

❖ Here is a very small sample data config file:

```
<?xml version=”1.0”?>
<Match version=”0.7.15”>
        <Config type=”Reference”>
```

```
        <Label>refType</Label>
        <Reference>
          <Key>RE</Key>
          <Value>Residential</Value>
        </Reference>
      </Config>

      <Data description=”Type”>
        <Source>
          <Filter type=”Reference” reference=”refType”>
            <Source>
              <Filter type=”Case” direction=”upper”>
                <Source>4</Source>
              </Filter>
            </Source>
          </Filter>
        </Source>
      </Data>
</Match>
```

- This file will have the 4th column of the data converted to upper case, then have it compared to the reference list to convert it. So after the upper case filter, if the value is RE, then the reference filter converts that to Residential.

Optional Configuration

The optional configuration is only needed for certain situations, and some of these situations will be described for these files. (Filters_Custom.pl and RealFeatures.pl.)

Filters_Custom.pl Configuration

This file is only needed if you want to add your own custom filters to RealParse. Please use this file instead of modifying the Filters_Base.pl file that is located with the RealParse script. Some rules you must follow to create your own Filter_Custom.pl file:

- Always end the file with a line that contains only, "1;". (That's the number 1 and a semi-colon) This is needed by Perl in some situations, so please put it at the end of each of your filter files.
- Each filter subroutine must end in the word, "Filter". This is to keep all the Filter subroutines separate from any other type of subroutines that RealParse may use. When specifying a filter in the Layout.xml file, you do not include the word, "Filter", in the filter's name, just the part before the word, "Filter". For example, the Split filter's subroutine is labeled, "SplitFilter", but when specifying that filter in the Layout.xml file, you refer to it as simply, "Split".
- Use "my" with all of your filter variables to make sure that variables don't bleed over into anything else.
- After you have created your Filter_Custom.pl file, remember to update your RealConfig.xml file and update the <Filter_Include> block by placing a new <Filter></Filter> line that contains the full path and name of your new filter file.

Here is a small sample filter, which we'll explain in more detail later, that you can use as a base for creating new filters:

```
1:      sub DumbFilter() {
2:         my($filterblock) = @_;
3:         my $source = $filterblock->{'Source'};
4:         my $value1 = $filterblock->{'value1'};
5:         my $value2 = $filterblock->{'value2'};
6:         my $data;
7:         if (ref($source) eq 'ARRAY') { # multiple sources
8:            foreach (@$source) {
9:               $data = &parseSource($_); # let &parseSource figure out what the value is
10:              $data = $data; # this is a dumb filter, so we do dumb things
11:           }
12:        } else { # just one source... pass it off to &parseSource to figure out what value is
13:           $data = &parseSource($source);
14:           $data = $data; # this is a dumb filter, so we do dumb things
15:        }
16:        return $data;
17:     }
```

Using the line numbers in the example, here is a line-by-line explanation of this filter:

1. This is the start of the subroutine and the only thing you need to change with this is the word, "Dumb". Just replace that word with the name of your filter.
2. This line gets the XML data block of the correct <Filter>. Just think of $filterblock as being the "<Filter>...</Filter>" data. (This would include any other sources or filters within this <Filter>.)
3. The $source variable is XML data block of the "<Source>...</Source>" that is within this filter. It is not required to be set if your filter works like the StaticValue filter where it ignores any Sources within it.
4. This line, as well as line 5, will change from filter to filter and possibly grow to many lines or be completely removed. They are simply variable lines that get variables that were set in the <Filter> element. In the example, line 4 gets the variable named, "value1", and line 5 gets, "value2". The

<Filter> line might look similar to this:
<Filter type="DumbFilter" value1="abc" value2="123">

5. Look at line 4 for more detail.
6. This creates an empty $data variable to place the raw data in so that we can modify it.
7. If there is more than one source block (<Source>...</Source>) then the $source variable will be an array reference. If the $source is an array reference then we must walk through it and look at each <Source>.
8. Foreach loop that loops through the array reference.
9. This line gets the data from the current <Source> block. It gets the data by calling on &parseSource and passing it the reference to the <Source> block that it wants to retrieve data for. The &parseSource subroutine will look at the <Source> and handle any <Filter> elements inside the <Source> block. This can be a recursive process, so it's best to leave this up to &parseSource to handle for you.
10. In this example filter, we basically do nothing with the data from the <Source> block. This is basically where you would place your code that would modify the data.
11. End the foreach loop.
12. If the $source variable was not an array reference then it was just a single <Source> block and we don't have to step through any array.
13. Pass the $source reference off to the &parseSource subroutine to get the data from the <Source> block.
14. In this example filter, we basically do nothing with the data from the <Source> block. This is basically where you would place your code that would modify the data.
15. End the if block.
16. Return the $data variable back as the result of this filter.
17. End filter subroutine.

You can place multiple filters within the same file, but remember to end the file with, "1;". If you need to look at more examples, please take a look at the Filters_Base.pl file where the standard filters are located.

RealFeatures.pl Configuration

The RealFeatures.pl file is used for parsing the Features data from a real estate feed. There are only a few things you need to know before creating your own custom version of this file for your data feeds:

- The variable that references the <Reference_List> in the RealConfig.xml file is:
  ```
  $config->{'Custom'}->{'Reference_List'}
  ```
  Use this variable to open your Reference List file that will normally contain some sort of mapping of feature values from the raw feed to actual database values.
- Always name your subroutine, "listFeatures", so that RealParse knows how to call it. You may have other subroutines within your file but the main subroutine that will be called by RealParse will be the &listFeatures.
  - When RealParse calls &listFeatures, it will pass the raw feature data from the data feed to this subroutine. No other variables are passed.
- As your code parses the feature data, append all the valid results into one long string and separate each feature with a "\n" so that the string would appear to be one feature per line. Once all parsing is complete, return this string back to RealParse. No other values should be returned. If your code cannot determine any valid features, simply return an empty string.
- To print any information to the screen or logfile, use:
  ```
  print $logOutput "text";
  ```
- If you need to view some examples, please look at the two versions provided with RealParse: RealFeatures_1d.pl and RealFeatures_2d.pl.
- The file should always end with a line that has "1;". (That's a 1 and a semi-colon)

Other Installation Requirements

RealParse needs no web server, database, or external program other than Perl and the Perl modules: XML::Parser and XML::Simple. *Perl and the modules may have other dependencies.*

Data Description Values

There are many data description values that RealParse looks for. This version of RealParse is static in the sense that it will only look for these data descriptions in the Layout.xml file, and nothing else. So make sure you are using a valid data description for each of your <Data> blocks in the Layout.xml file. Here is a list of valid descriptions:

MLS	Price	Class	Active/Inactive(Y/N)	
Agent ID	Agent Full Name	Agent First Name	Agent Middle Name	Agent Last Name
Agent Day Phone	Agent Night Phone	Agent Address 1	Agent Address2	Agent City
Agent State	Agent Zip	Agent Country	Agent E-Mail	Agent WWW
Agent Status				
Broker ID	Broker Full Name	Broker First Name	Broker Middle Name	Broker Last Name
Broker Day Phone	Broker Night Phone	Broker Address 1	Broker Address2	Broker City
Broker State	Broker Zip	Broker Country	Broker E-Mail	Broker WWW
Broker Status				
Business Name	Address 1	Street Number	Street Direction	Street Name
Street Type	Address 2	City	State	Zip
Neighborhood	Subdivision	Sale/Rent	Total Bedrooms	Total Baths
Total Full Baths	Total Half Baths	Total ¾ Baths	Image URL	Garage Capacity
Garage Type	Carport Capacity	Carport Type	Garage	Basement
Style	Fireplace	Type	Year Built	Age
Lot Dimensions	Lot Size	Elementary	Junior High	Senior High
Master Bedroom Size	2nd Bedroom Size	3rd Bedroom Size	4th Bedroom Size	5th Bedroom Size
6th Bedroom Size	Dining Size	Family Room Size	Game Room Size	Great Room Size
Kitchen Size	Laundry Room Size	Living Room Size	Rec Room Size	Utility Room Size
Breakfast Size	Den Size			
Master Bedroom Level	2nd Bedroom Level	3rd Bedroom Level	4th Bedroom Level	5th Bedroom Level
6th Bedroom Level	Dining Level	Family Room Level	Game Room Level	Great Room Level
Kitchen Level	Laundry Room Level	Living Room Level	Rec Room Level	Utility Room Level
Breakfast Level	Den Level			
HOA Fees	Monthly HOA Fees	Annual HOA Fees	Taxes	General Taxes
Special Taxes	Balance Assessment	Average Electric	Average Gas	Total SqFt

Total SqFt Above Ground	Total SqFt Below Ground	Main Floor SqFt	Upper Floor SqFt	Lower Floor SqFt
Basement Floor SqFt	Auction Date	Auction Time	Auction Location	Number of Acres
Units Vacant	Units Occupied	Total Units	Stories	Ceiling Height
Current Use	Listing URL	Virtual Tour URL	Home Audio URL	Home Video URL
House Depth	House Width	Community ID	Community Name	Community Address
Community City	Community State	Community Zip	Community Market ID	Community Market
Move In Date	Contact Name	Contact Phone	Contact E-Mail	Plan Type
House Dimension	Remarks	Features		

Administrator Maintenance

Configuration Files

Please refer to the Configuration section.

Error Logs

The logs are generated as RealParse runs. The location and the name for the log file are set in the RealConfig.xml file. Please look at the Configuration section and look for the <Log_Filename> element description for more detail on setting up the log files.

Recommended Cron Setup

To keep things simple, a single cron entry that runs late at night is the most convenient way to cron the RealParse. The cron would run a single script that calls on RealParse and passing it a configuration file of a property using the -c flag. Here is a sample line from my script that is run nightly:

perl /opt/MDW_DD/realparse/RealParse.pl -c /opt/MDW_DD/conf/mlsyork/RealConfig_residential.xml

External Data Dependencies

Each property level installation of RealParse requires a real estate data feed.

Program Recovery Procedures

No recovery necessary for a failed run, simply have them upload new data and run RealParse again. If it still fails to output the correct data, then check your config files (the layout XML file specifically) and if those are correct, check the data feed. Nine times out of ten the data feed will be corrupt if there has been successful parsings in the past.

Troubleshooting Guide

There will be a FAQ generated in the near future to help with some common problems, but until then, you'll have to contact Morris Support Services for any problems you may run across if you cannot debug them yourself. Please look at the Support Information section of this document for contacting Morris Support.

There are a few things you can do to make life a bit easier when debugging a problem with RealParse, such as command line flags. You can pass these flags to RealParse when you execute it, which will change logging functionality to display more debug information and other similar things so you can easily see whets happening. To get the latest list of these flags you can simply run RealParse without a configuration file and it will display its usage information. Here are the command line flags as of this writing:

- (-c <file> | --c=<file>) : This is a required flag that specifies the RealConfig.xml file you want RealParse to use when parsing. As with all variables, it can be passed with a single dash and a space between flag and value, or a double dash with an equal sign between flag and value.
- [--show_errors] : Passing this flag will print all errors to the logfile/screen if errors were disabled by the RealConfig.xml file.
- [--show_debug=<1..100>] : This flag will overwrite the debug level that is set in the RealConfig.xml file and print debug messages to the logfile/screen. Some debug messages only display when the debug level is high enough, so if you want to see all messages, set this to 100, but if you only want to see some, start with 1 and work your way up.
- [--disable_log] : This flag will ignore all log file settings in the RealConfig.xml and print all debug and error information to the screen instead.

Support Information

Morris Digital Works (MDW), a division of Morris Communications Co. (MCC), provides tools, technologies, consulting, and Web development services to Morris newspapers and external clients. MDW award winning technologies include: world-class hosting facilities, robust content management software, high performance application tools, site enhancement tools, and comprehensive classified and display classified technology. Founded in 1995, MDW has over 100 employees with offices in Topeka, KS, Joplin, MO, New York, NY, and is headquartered in Augusta, GA. An additional 250 MCC employees also participate in our Internet business and report directly to newspapers, magazines, book publishing, and other internal organizations.

If you are experiencing problems with any Morris Digital Works product, please contact customer support at (706) 828-2955.