InfoLinker PLUS

An Automated Bookmark and Linking Plug-in for Adobe Acrobat

September 2014

DocMaestro, LLC.
5180 Venetian Blvd
Saint Petersburg, FL 33703

Table of Contents

List of Figures and Tables

Chapter 1: The InfoLinker Plus System

InfoLinker Plus Application Overview

InfoLinker Plus is an *Automated Bookmark and Linking Plug-in* for Adobe Acrobat. This is the newest in the series of Adobe Acrobat tools from DocMaestro LLC, Inc. ***InfoLinker Plus*** is a 32-bit application that runs on 32-bit or 64-bit operating systems and is compatible with Windows 7 & 8.1 and Adobe Acrobat 8, 9, & X (10).

InfoLinker Plus builds bookmarks and text links in PDF documents based on the text and structure of the document. Table of contents, titles, indices and references are easily linked using a user-constructed set of rules. External references can also be linked by rules. Page geometry, font details, phrase structure, text location, and a number of other common content discriminators are used to build the bookmarks and links in a document. Every occurrence of "see figure..." or "in table…" can be linked to the referenced item in the document. The figure or table can also be accessed from a bookmark created by ***InfoLinker Plus***. One set of ***InfoLinker Plus*** rules can automatically re-build the bookmarks and links in the document set every time the document is published or updated.

Rules Overview

InfoLinker Plus uses a Graphical User Interface (GUI) Rules Editor that is accessed from the Acrobat menus. The Rules Editor allows the user to build rules and to interactively select parameters for each rule. A rule can be applied to the complete document or a range of pages and is one of these basic types.

Destination

A destination is an object or document page view that is the target of a source link or bookmark. Any number of links or bookmarks can be linked to a destination. ***InfoLinker Plus*** creates Bookmarks and Name Destinations for all destination links.

Source

A source link is a reader selectable connection to a destination view. These links are connected to bookmark destinations that are created by ***InfoLinker Plus***.

Any

The "Any" rule can create results that are either a source or destination. The first occurring match in a document for a rule type "Any" is declared the destination and all subsequent matches are declared sources that link to the first occurrence.

Name

This rule type creates links to existing Name Destinations in other PDF documents. (Bookmark titles are one example of Name Destination.) Links in one document connect directly to a Name Destination view in another document. If the name is not found in the destination document then the link defaults to the first page of the destination document.

External

An External rule creates links to other programs and file types within the target system or network. A link to a video or spreadsheet application is an example of an external type link.

URL

A URL link type creates links to an http, https, email, or ftp paths.

Tree

A Tree rule is used to build nested bookmarks in a PDF document. A parent tree name is defined and a regular expression is used to determine which bookmark names will be nested under that tree.

Comment

A Comment rule can be used to help the rule writer organize and document the rule set as required. Comment rules have no effect on processing.

Manual

Manual source links can be defined in the rules. The zone coordinates determine the display of the defined page. The name of the source and its rename are defined in the rule to determine what destination the source link will link to. This rule type is useful for creating links from locations that have no underlying or associative text.

Bookmark

A manual destination / bookmark can be described in the rules using this rule type. The zone coordinates and page definition determine where the destination view will be. The name and rename define the name of the bookmark and the rename character pattern is used for source links to connect to the destination. This rule type is useful for creating links to locations that have no underlying or associative text.

Include

This rule type allows another document's existing rule set to be included with the current document's rule set during processing. The DocID of the existing document is placed in the RegExp field of the rule. Local symbols are used.

Skip On and Skip Off

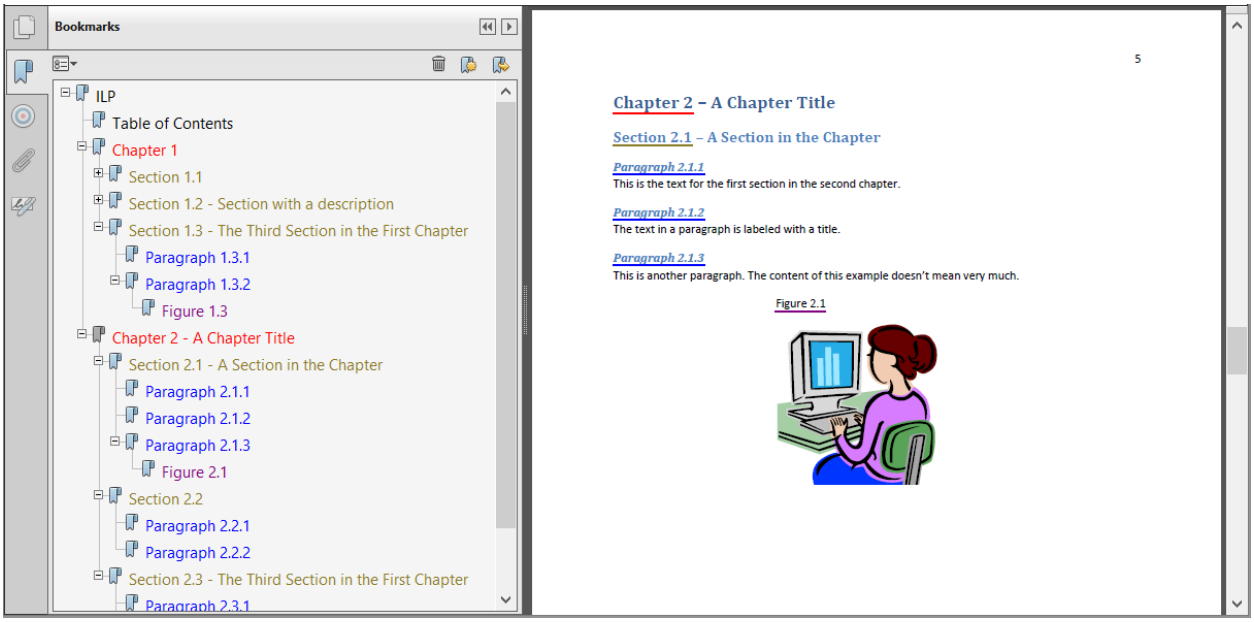
These two rule types are not yet implement. Working in conjunction, these rule types will allow sections of rules to be ignored during processing. All rules between the Skip On and Skip Off boundaries will be treated as comments. Local symbols will be used.

1 Root, 2 Root, and 3 Root

These three rule types are similar. These rule types define levels of bookmarks in a tree hierarchy. "1 Root" determines the lowest level of a tree structure. When a destination matches the rename pattern, that matching name becomes the default bookmark for all subsequent bookmarks that don't match other tree names. All of the subsequent bookmarks become children of that "1 Root". If a subsequent bookmark matches the "1 Root" pattern, a new root bookmark is created and becomes the default bookmark parent. The "2 Root" and "3 Root" rules build branches in the tree structures. This functionality is helpful in structured documents that have Chapter, Section and Paragraph structures.

The figure 1 screenshot is an example bookmark tree structure for a document using the three levels of the "1 Root, 2 Root, 3 Root" rules.

Figure 1. Structured Bookmark Tree Example.

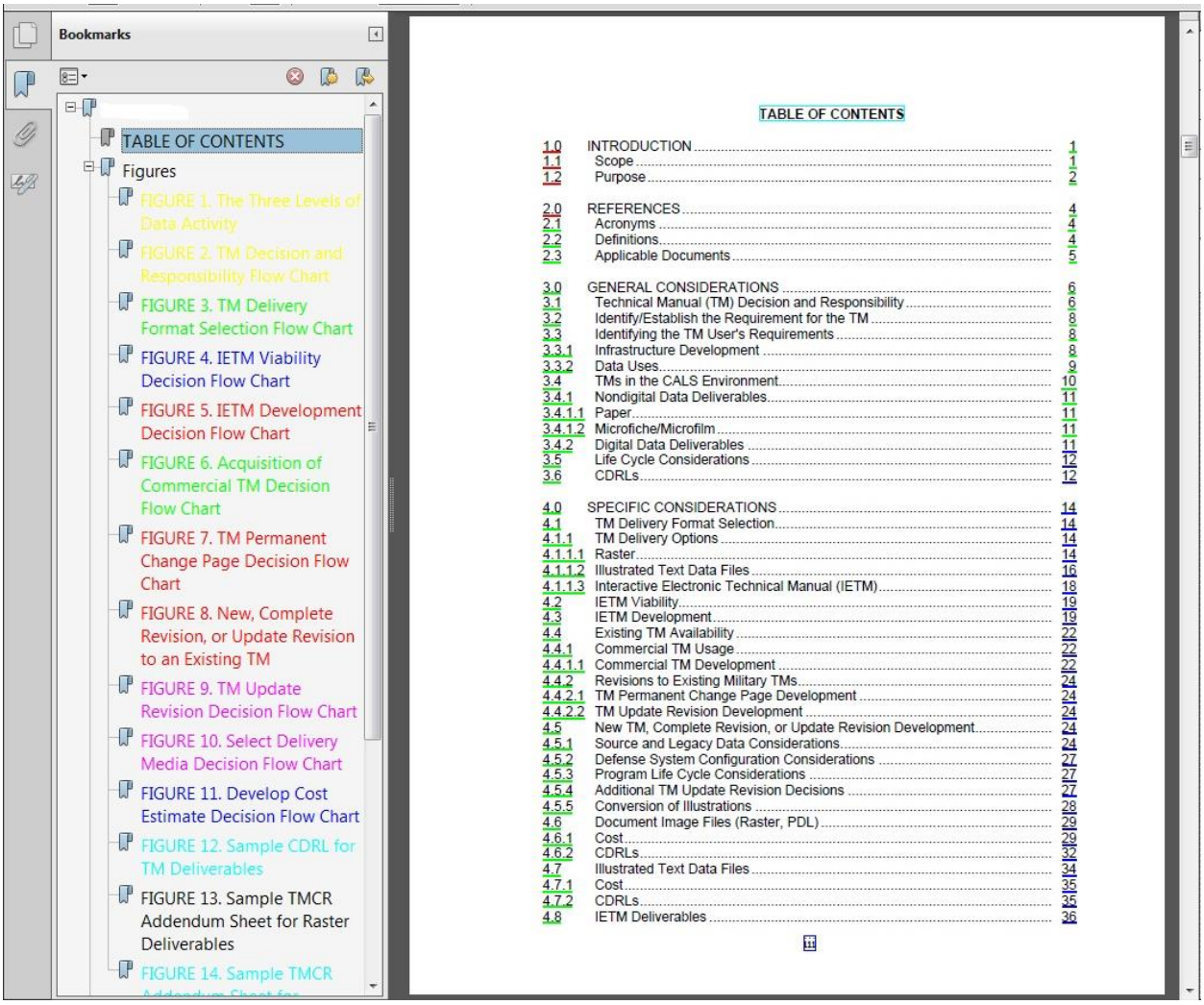


Basic Processing

The ***InfoLinker Plus*** process applies a set of rules to a document's contents and assists in the building of interactive features like annotations and bookmarks that provide end-users easier access to the contents of the document. Some sample screens and descriptions of the ***InfoLinker Plus*** product are included on the following pages.

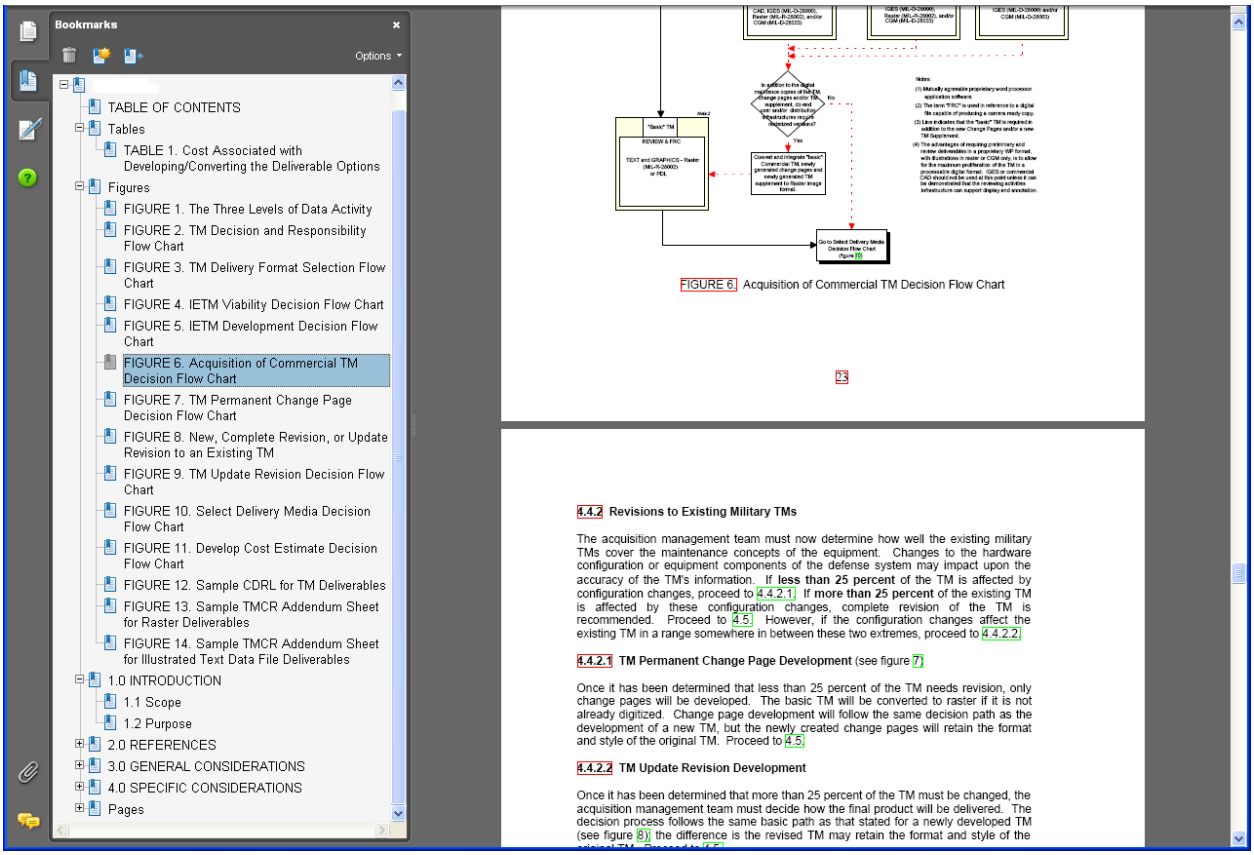
The figure 2 screenshot shows a sample PDF document that has a table of contents, figures, tables and paragraph and page numbering. ***InfoLinker Plus*** was used to build all of the bookmarks and document links that are used to improve document navigation. In addition to standard black bookmarks and boxed links, ***InfoLinker Plus*** includes option settings for color bookmarks and underline links.

Figure 2. A Document Sample Using InfoLinker Plus Navigation.



In figure 3, the green boxes are around links that connect to matching red boxes throughout the document. Default link colors are set in the program preferences and may also be modified using individual rule parameters.

Figure 3. A Source and Destination Example.



Manually linking and bookmarking this document in Acrobat would take hours. It takes significantly less time to construct the rules and apply them using ***InfoLinker Plus***. But the biggest savings using ***InfoLinker Plus*** is when the same (or a similar) document requires reprocessing. The originally constructed rules set can be applied even if a paragraph is added, a figure is changed, or content moved to a different page. ***InfoLinker Plus*** rebuilds the newly published document using the same set of rules and all of the links and bookmarks can be regenerated in minutes.

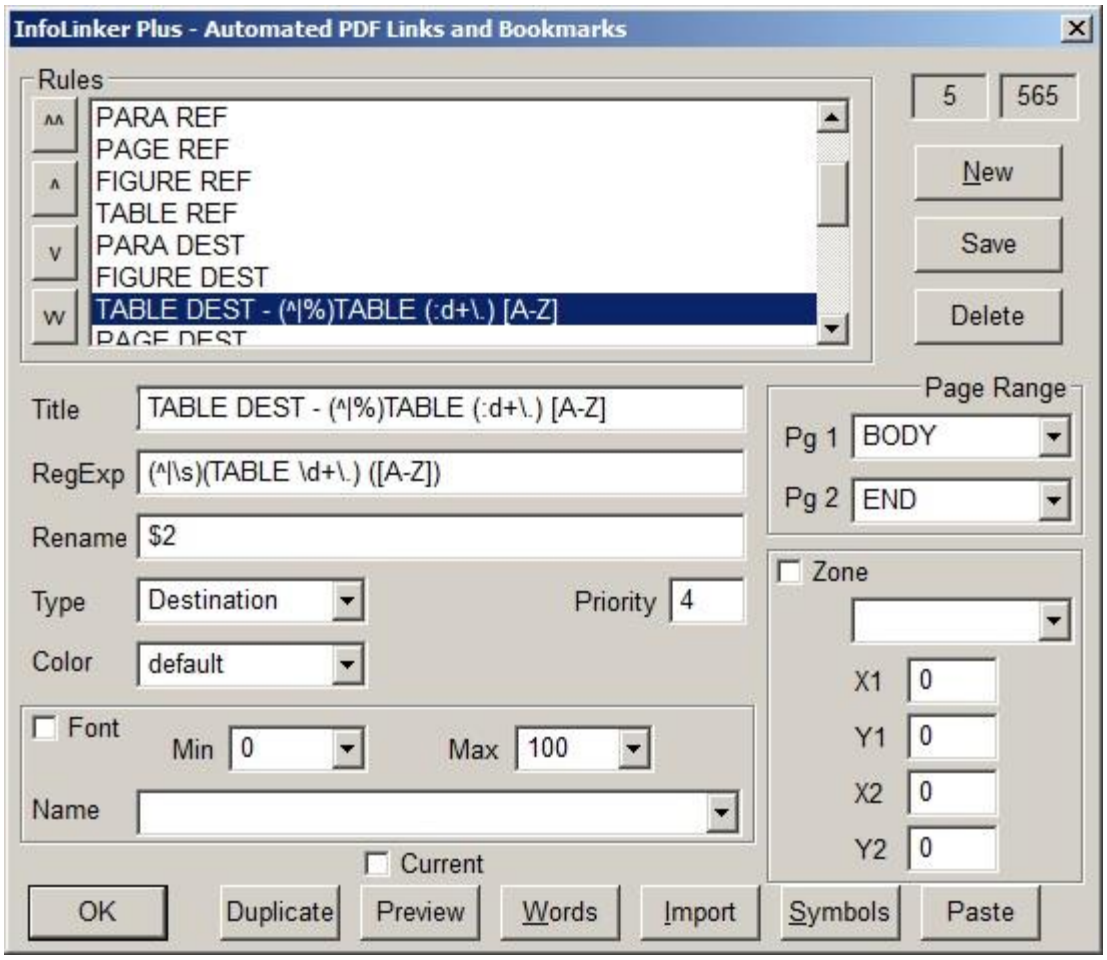
InfoLinker Plus Components and Tools

GUI Rules Editor

InfoLinker Plus use *regular expressions* rules similar to those used in InfoLinker. The colon string patterns, phrase break (%), and most other functions can still be used. However, there are a number of additions and a changes in the expression syntax. The rules now include options for tab, line feed, and form feed. These functions allow you to narrow the potential hits and improve accuracy in the results.

The dialog window in figure 4 is used to create and edit the rules that define the bookmarks and links in the document.

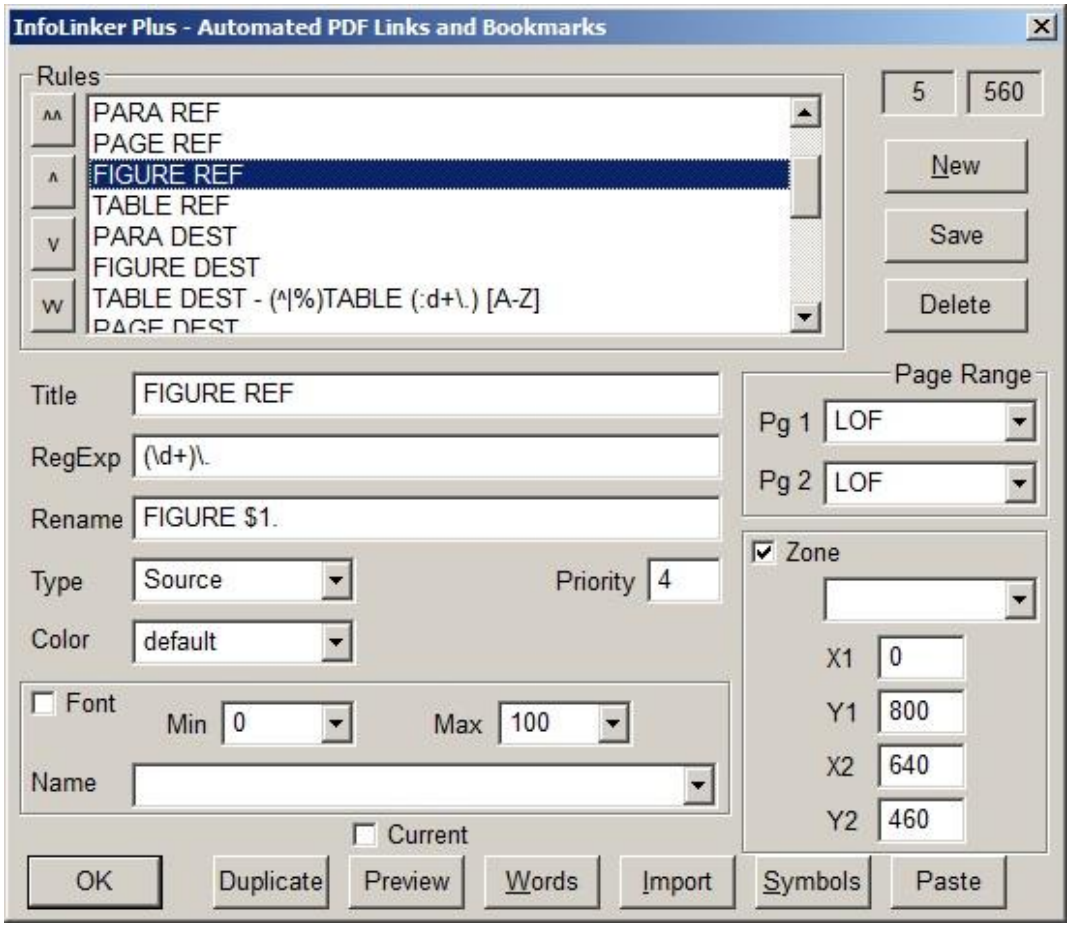
Figure 4. The Rules Editor Dialog.



In the example above, the \d in the RegExp line is the same as :d (or [0-9]) that are used in InfoLinker rules. Rename syntax in ***InfoLinker Plus*** is also different from InfoLinker. In the Rename field, the $ followed by a number represents the matching set number. Earlier versions of InfoLinker used the [*n*] pattern for renaming matching. That pattern is not part of the evolved standard regular expression syntax. The ***InfoLinker Plus*** syntax has been adjusted to stay current with the evolved standard and (when possible) still allows the older expression syntax. A rule import function is included that imports older rules and modifies their syntax for compatibility with the new standard (see Chapter 4).

The rule in figure 5 (below) uses page geometry as well as font information to specify a destination in a document. This data can be populated by selecting text in a PDF and pressing the "Paste" button. This inserts the "Zone" and "Font" information as well as the selected text into the appropriate fields.

Figure 5. A Rule for Linking to Figures.



The rule Types include: Destination, Source, Any, External, URL, NAME, Tree, Comment, Manual, Bookmark, Include, 1 Root, 2 Root, and 3 Root. (See figure 18)

The Font "Name" dropdown populates with all of the available fonts in the current document. The Min and Max are useful in differentiating document body text from paragraph and chapter titles. Min and Max are currently only implemented in conjunction with a font name.

The "Color" dropdown lists the available link/bookmark colors. If a rule color is selected, then all links and bookmarks that result from this rule will be displayed in the selected color instead of the default color value. This adds highlights to the document navigation and can also be very helpful in understanding the impact of a particular rule in the resulting links.

The "Preview" function is helpful in testing a rule prior to applying it to the complete document. Preview applies the selected rule and builds preview link boxes and

bookmarks for the "Current" page or defined page range. Zones and link color settings are ignored by the "Preview" function. The link boxes are drawn in green around any matching text. The preview bookmarks ignore the renames and instead display the matching text (including predecessor/successor) of the defined links.

Several other tools are also included with the ***InfoLinker Plus*** plug-in. These tools can be accessed directly from the ILP application directory or from the Acrobat via the ILP Plug-Ins menu and screens. The Preferences program (**RegPref**) is used to set the application preferences. The Regular Expression program (**RegExp**) includes the compiler and linker functions that process the rules against the selected document. The PDF Words Viewer program (**Words**) displays all of the words from the current (or selected) PDF page. The Log program (**RegLog**) accesses the results for the Prep, Compile, and Link logs. The Rules program (**RegRules**) lists the rules for a loaded document and it can be used to copy, import, or save rules. Rules can be imported from text files such as InfoLinker rule (.rul) format files or from previously saved ***InfoLinker Plus*** (.cvs) files. Screenshots and brief descriptions follow.

InfoLinker Plus Matching Options

Case Insensitive Matching

There are a few things that can be added to a rule to change the rule result. If the first character in a regular expression of a rule has the tilde ("~") character, then the rule will ignore case sensitivity. For example an "A" will be the same as "a" when matching the expression to the contents of the target in the document.

Predecessor / Successor Matching

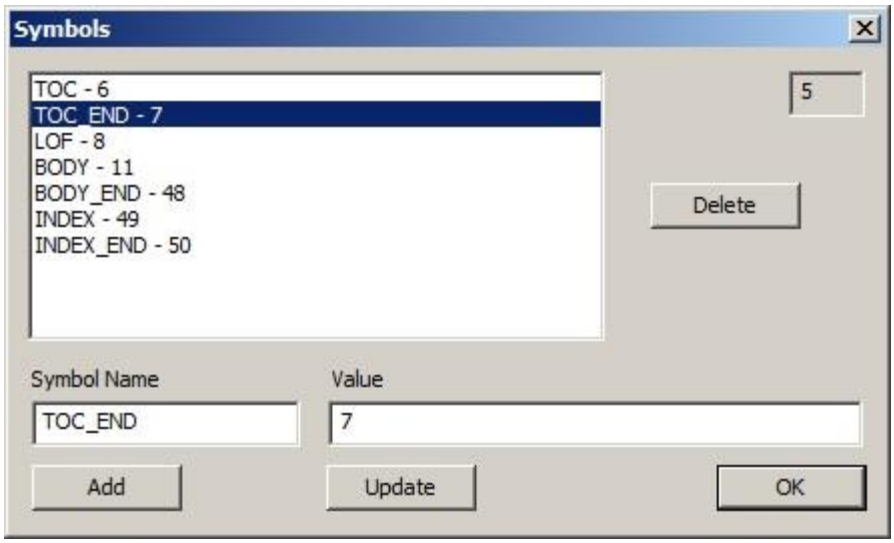
Another rule modification is in the use of predecessor and successor syntax. The "@@" characters are used to break a rule in to component phrases. All content before the first occurrence of "@@" is defined as the predecessor phrase. If used, all content after a second "@@" is defined as the successor portion of the expression. All content between the symbols defines the boundaries of the link box or underline. The entire regular expression (minus the "@@" symbols) must match strings in the document to be potential links from the rule. The predecessor part of the rule will not be highlighted if the string becomes a link. The successor part of the rule, if provided, will also not be highlighted as part of the link. This provides a rule writer with the ability to better qualify strings as links in documents without making the links highlight the entire matching pattern. Rename variables can be used with predecessor / successor matching in any part of the expression.

InfoLinker Plus Symbols

InfoLinker Plus maintains a set of symbols for each document that it processes. Symbols can used to define page names and zones. The values associated with these symbols are user specified. Page name symbols can be used in rules in place of the PDF page number. Zone name symbols store coordinate information and are useful when the same zone data is applied to more than one rule. The Symbols

dialog (figure 6) can be called from the *InfoLinker Plus* "ILPSymbol" menu in Acrobat or from the Symbol button on the Rules Editor. The user can add 2 main types of symbols in this dialog. Page names are symbol names that have integer values. All of the page name symbols will appear in the Rules Editor's page dialog drop down. This allows the user to define special pages that describe the organization of the document. The TOC, BODY, END_BODY, INDEX, APPENDIX, and any other page that may be used in the rules can be defined and updated in one place using the Symbols dialog.

Figure 6. The Symbols Dialog.

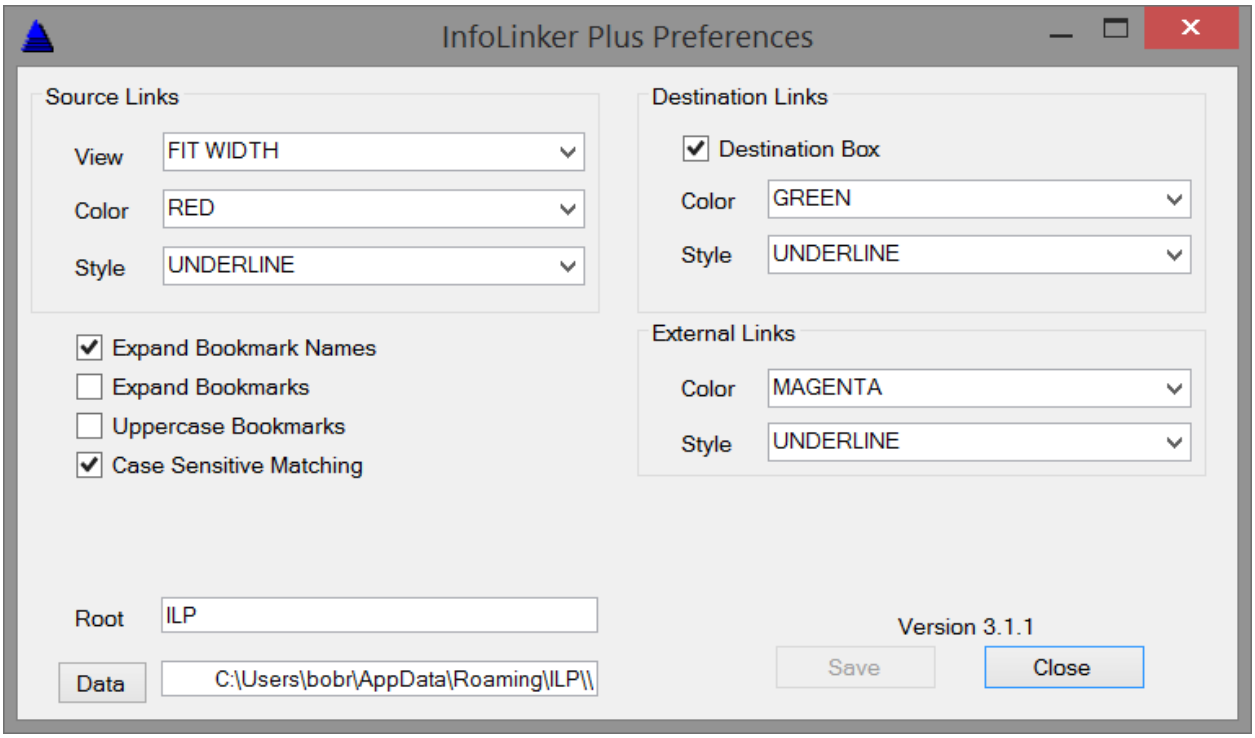


The other symbol type that is maintained in this symbol dialog is the names used in external type rule renaming. The user can use 2 character long symbol names that start with an "H" that can be referenced in rule renames. This limits the number to 36 different symbols per document. The symbol name is replaced by the string value that is entered in the symbol dialog. In the above example there are 2 symbols that can be used in the rename of rules that are "H1" and "H2".

InfoLinker Plus Preferences

There are a number of preferences that can be set to define how the bookmarks and links that are created and displayed. This dialog is accessed using the ILP Preferences menu item. A screenshot of the Preferences dialog (figure 7) is shown below.

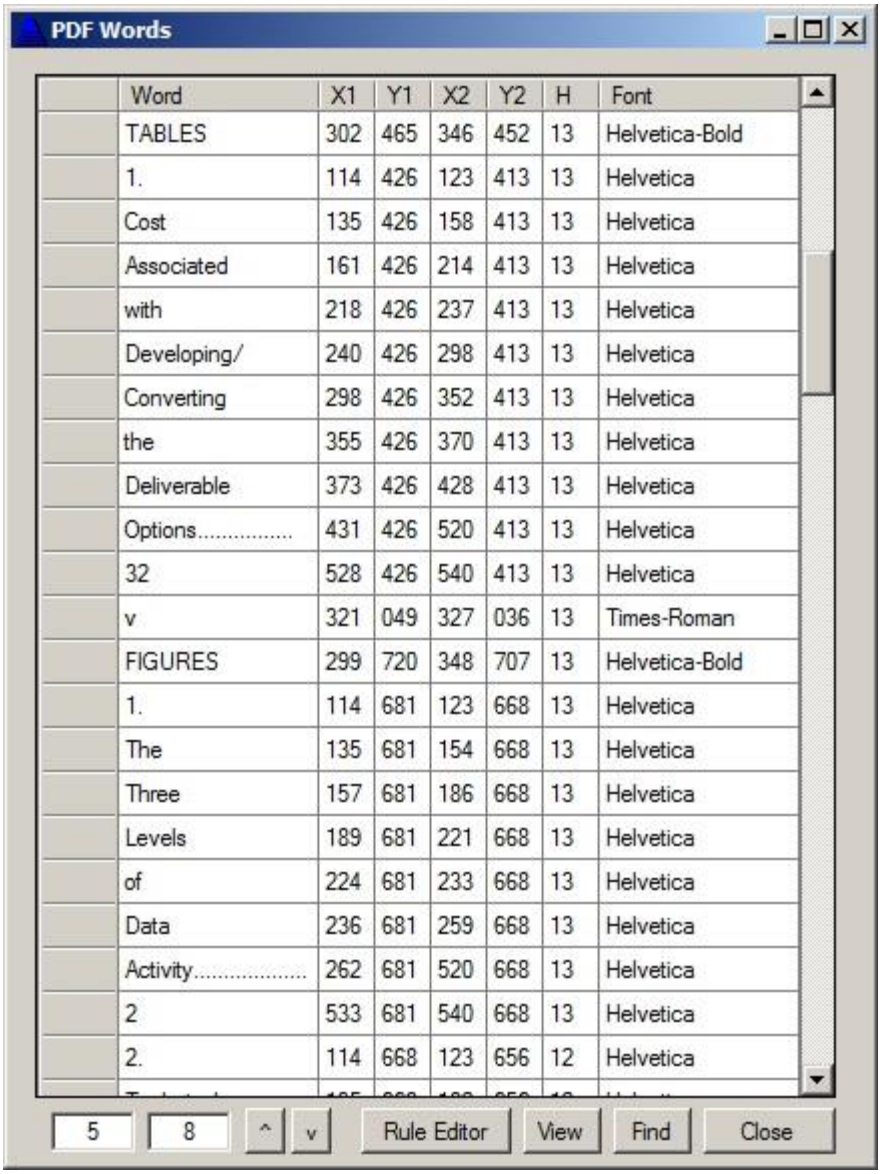
Figure 7. The Preferences Dialog.



PDF Words Viewer

The "Words" button on the Rules Editor dialog displays a window (figure 8) with all of the words on the current page. Each word displays its PDF coordinates and font information. The dialog displays the current Document ID ('5') and page number ('8'); allows toggle using Rules Editor button, page forward and page back using the arrows, jump to page by alternating page number and pressing View button, and Find words on current page.

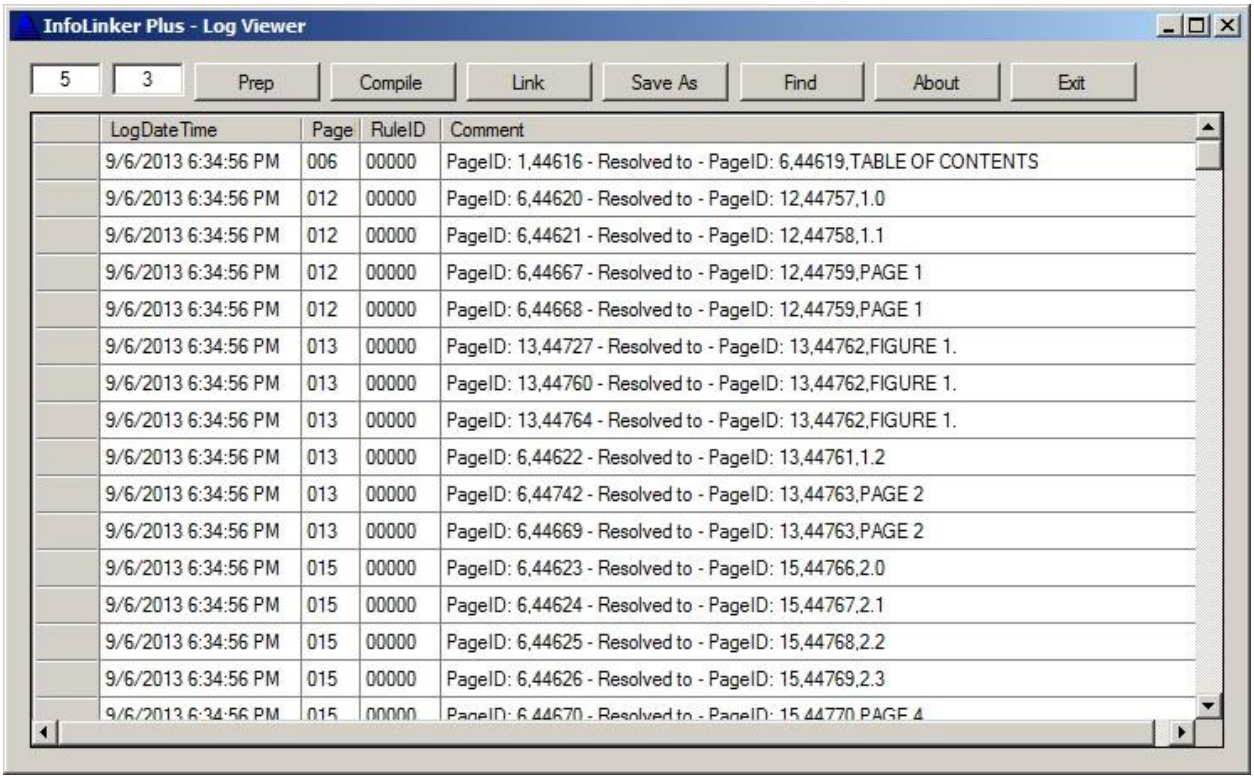
Figure 8. The Words Dialog.



Log Viewer

There is a log function with ***InfoLinker Plus*** that lists the detailed compilation of the rules against each page of the document. The log (figure 9) shows which rules hit on which pages and the associated results. Clicking on the column header, allows the results to be sorted by "Time", "PageID", "RuleID", and /or "Comment". The results can be saved to a text file (if desired) in any selected order.

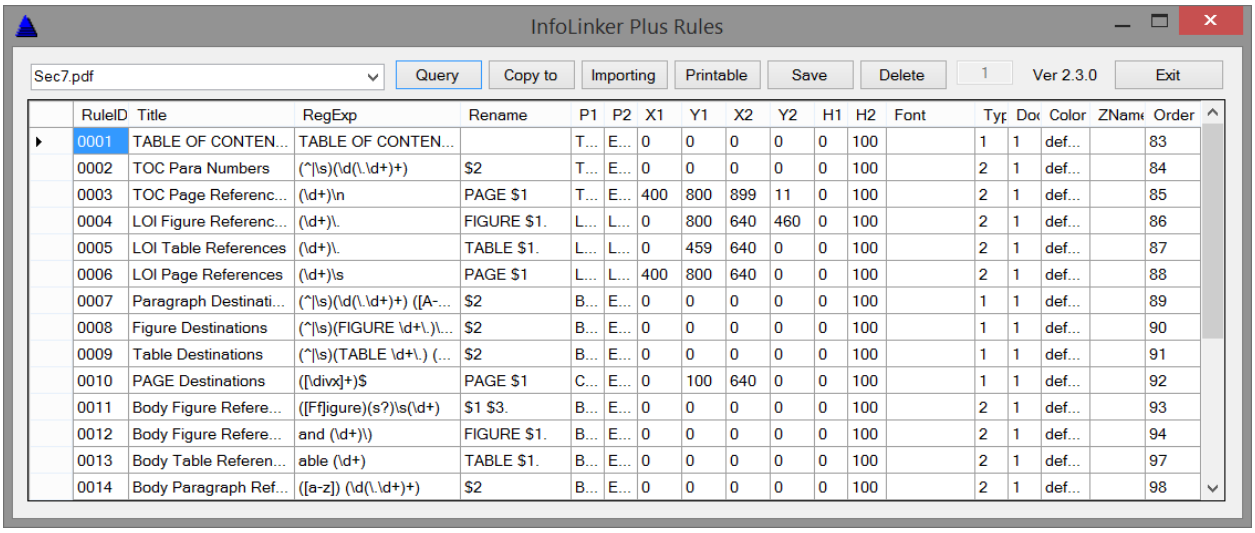
Figure 9. The Log Viewer.



Rules Manager and Importer

The RegRules program (figure 10) is a utility program for rule management. This program allows users of ***InfoLinker Plus*** to import existing *InfoLinker* rules into the ***InfoLinker Plus*** database rule structure. The RegRules program also allows a user to copy rules from one document to another, similar to using templates. RegRules is launched by pressing the RuleDB button on the Rules Editor (or by launching the RegRules.exe in the application directory).

Figure 10. The RegRules Dialog.



RuleID	Title	RegExp	Rename	P1	P2	X1	Y1	X2	Y2	H1	H2	Font	Typ	Doc	Color	ZName	Order
0001	TABLE OF CONTEN...	TABLE OF CONTEN...		T...	E...	0	0	0	0	0	100		1	1	def...		83
0002	TOC Para Numbers	(^\s)(\d(\.\d+)+)	$2	T...	E...	0	0	0	0	0	100		2	1	def...		84
0003	TOC Page Referenc...	(\d+)\n	PAGE $1	T...	E...	400	800	899	11	0	100		2	1	def...		85
0004	LOI Figure Referenc...	(\d+)\.	FIGURE $1.	L...	L...	0	800	640	460	0	100		2	1	def...		86
0005	LOI Table References	(\d+)\.	TABLE $1.	L...	L...	0	459	640	0	0	100		2	1	def...		87
0006	LOI Page References	(\d+)\s	PAGE $1	L...	L...	400	800	640	0	0	100		2	1	def...		88
0007	Paragraph Destinati...	(^\s)(\d(\.\d+)+) ([A-...	$2	B...	E...	0	0	0	0	0	100		1	1	def...		89
0008	Figure Destinations	(^\s)(FIGURE \d+\.)\...	$2	B...	E...	0	0	0	0	0	100		1	1	def...		90
0009	Table Destinations	(^\s)(TABLE \d+\.) (...	$2	B...	E...	0	0	0	0	0	100		1	1	def...		91
0010	PAGE Destinations	([\divx]+)$	PAGE $1	C...	E...	0	100	640	0	0	100		1	1	def...		92
0011	Body Figure Refere...	([Ff]igure)(s?)\s(\d+)	$1 $3.	B...	E...	0	0	0	0	0	100		2	1	def...		93
0012	Body Figure Refere...	and (\d+)\)	FIGURE $1.	B...	E...	0	0	0	0	0	100		2	1	def...		94
0013	Body Table Referen...	able (\d+)	TABLE $1.	B...	E...	0	0	0	0	0	100		2	1	def...		97
0014	Body Paragraph Ref...	([a-z]) (\d(\.\d+)+)	$2	B...	E...	0	0	0	0	0	100		2	1	def...		98

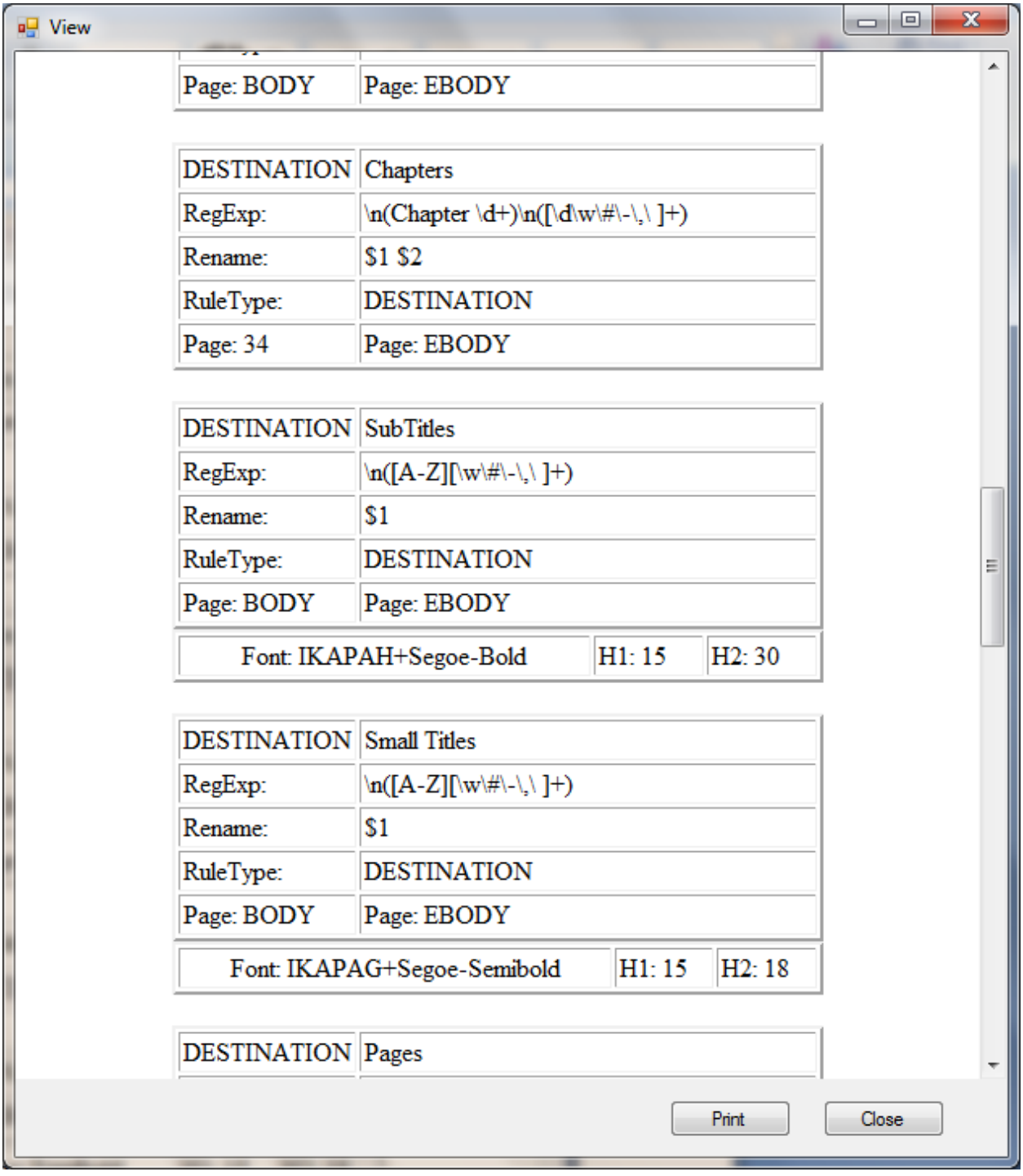
Importing existing InfoLinker rules can be accomplished using RegRules. There is a button on the top of the dialog called "Import". A user presses the Import button and either selects an existing document from a list or adds a new document to the list and selects the new document. After a document is selected a file open dialog is displayed that the user uses to browse to the rule file (usually a .rul suffix) and they select the appropriate rule file. The import function will add rules, zones, and symbols to the document database while translating the rule file to the ***InfoLinker Plus*** structure.

Another function of RegRules is to format the rules of a document into a printable format. Once a set of rules is displayed in the dialog listing, the user can press the "Printable" button. This will open a browser window and display a tabular listing of the rules for the selected document. An example is displayed in figure 11 below.

The "Print" button sends the printable view of the rules to the default printer on the workstation.

The "Save" button allows you to save the current rule set to an Excel (.csv) file. The saved .csv file can be later imported into similar documents on the same workstation or to other workstations. The .csv can be emailed and is useful in requesting support assistance from other users or support personnel.

Figure 11. Printable Rule Listing.



InfoLinker Plus Process

The ***InfoLinker Plus*** process consists of the steps listed in table 1. A more complete description of each step follows the table.

Table 1. InfoLinker Plus Process.

Step	Name
(1)	Add Document to Database
(2)	Prepare Document
(3)	Rule Writing and Editing
(4) Optional	Remove Previous Links
(5)	Process Document (Compile and Build)
(6)	Update Document

Step 1 - Add Document. The first step is entering a PDF document into the database using the Document Database dialog. This dialog can be launched from the Rules icon or from the ILP Prepare or ILP Rules menu items.

Step 2 – Prepare Document. This "Prepare" step copies all the words and related data from the PDF document and inserts them into the document database. The associated word data includes the page number, coordinate position, font size, font name, and word sequence. This data is used by ***InfoLinker Plus*** during creation of links and bookmarks.

Step 3 – Rules Writing and Editing. The rule writer team member then builds rules using either the Rules Editor or the Rules Import utility. The rules are used by ***InfoLinker Plus*** to identify potential links and bookmarks. This step may also require that symbols get added to the database. Symbols are names for pages and names for zones that are used in the rule writing step.

Step 4 – (Optional) Remove Previous Links. The "Remove All Extras" plug-in can be launched to remove any existing links, bookmarks, and name destinations from the document. To avoid duplicate links and bookmarks, this plug-in should be run either before the ***InfoLinker Plus*** process or update steps. Another alternative to remove unneeded annotations is to use the Link Manager menu item to remove specific link types and annotations from the document.

Step 5 – Process Document. This process can be launched from the ILP Process Icon or menu. This step includes two critical components:

- Compile Potential Links. The rules are processed against the words in the PDF one rule at a time in priority order. This step creates a list of all the potential links and bookmarks in the document. The matching strings of the Regular Expressions (RegExp) become the names of the items in the list. If there is a rename pattern then that replaces the name of the item in this list. This step is sometimes referred to as compiling the document. At this point none of the items are marked as part of a link or bookmark.

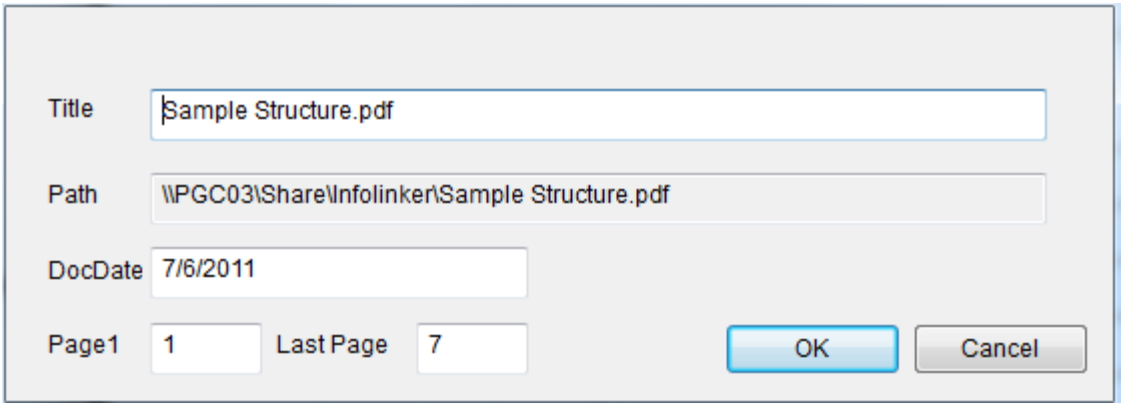
- Build Links and Bookmarks. The Link stage processes the list of potential items. The linking step takes each of the items created by "Destination" rules and defines bookmarks for the items. Items that were created from "Any" rules are processed next. If there is more than one in the list from an "Any" rule then the first in the document is defined as a destination and all of the others are defined as source links that are pointing to the first one. The source items in the list are then compared to each of the destinations and if they match a destination then they are marked as source links and are paired to the destination. The other link types are processed one at a time and determined if they create items. Next, the bookmark items are again processed to see if they are part of a bookmark tree.

Step 6 – Update Document. The last stage of the ***InfoLinker Plus*** process is referred to as the "Update" step. This step takes the tree structured bookmarks and inserts them into the PDF and builds, if necessary, a highlight for the destination. The source links from the link stage are then found in the PDF and based on their color and link style a link is built that takes a viewer from the source link to the view of the bookmark it has been matched to. Any external links are added in the last step of the update.

Document Control

There is a new document dialog (figure 12) that a user completes the first time a document is opened in ***InfoLinker Plus***. The user is allowed to edit the name. By default the name is the path to the opened document. ***InfoLinker Plus*** maintains the date and time of preparation and processing of the document as part of the database. The database maintains the rules for the document, the contents of the document, and any hotspot items that result from processing the document using the rule compiler.

Figure 12. The New Document Dialog.



Database

The database used in ***InfoLinker Plus*** is the Microsoft Access 2010 database. Microsoft provides free database tools that ***InfoLinker Plus*** uses for access to the database system. A full license to Access 2010 is not required to run the

InfoLinker Plus program. The user does not have direct access to the database contents. The database is password protected and all data access necessary for use should be available using the ***InfoLinker Plus*** products and related tools.

Operating Environment and User Rights

InfoLinker Plus is a combination of an Acrobat Plug-in that runs inside Acrobat and a set of tools that are used to process user documents. Currently, the ***InfoLinker Plus*** system has been tested in the Windows 7 & 8 (x86) operating system and the Windows 7 & 8 (x64) operating system. The product suite has been tested to work with Acrobat 8, 9, X (10) and XI (11). ***InfoLinker Plus*** uses registry settings to establish the location of the different parts of the program set. These values are set at installation time. The path to the database and support programs needs to have full access rights in that directory. The recommended installation location for the database is C:\Users\{user}\AppData\Roaming\ILP. The location for the support programs is C:\{Program Files}\DOCMAESTRO\ILP. The system can be configured in other ways if required. Acrobat does not need to be run in Administrator mode. There are no references to the Windows directory or changes to the Windows directory required to operate ***InfoLinker Plus***.

x64 vs x86 Operating System

Adobe Acrobat's current developer's kit has dictated the architecture of ***InfoLinker Plus***. Acrobat is currently only available as a 32-bit program. It is compatible with and installed in 64-bit windows Operating Systems as a 32-bit program. The registry values for Acrobat are in the WOW6432 registry of 64-bit systems, and that is where ***InfoLinker Plus*** registry values will be placed. The ***InfoLinker Plus*** support programs and database could all be in 64-bit format now, but until Adobe Acrobat moves to the 64-bit these must remain 32-bit applications.

Installation

During installation, the ***InfoLinker Plus*** plug-in (*ILP.api*) is copied to the current Adobe Acrobat plug-in directory. For Acrobat 9 this normally is C:\Program Files\Adobe\Acrobat 9.0\Acrobat\Plug-ins on the Windows 7 (x86) operating system. For Acrobat X on Windows 7 (x64) the location for ILP.api is C:\Program Files (x86)\Adobe\Acrobat 10.0\Acrobat\plug_ins. The support programs should be located in C:\{Program Files}\DOCMAESTRO\ILP and the ILP.accdb file should be located in C:\Users\{user}\AppData\Roaming\ILP. If this directory does not exist, it will need to be created with appropriate access rights. There are 2 registry keys name/values: (the values lists below are for 64-bit operating systems)

HKEY_LOCAL_MACHINE\SOFTWARE Wow6432Node**DOCMAESTRO\ILP**
Name: DBPath Type: REG_SZ
Value: C:\Users\{username}\AppData\Roaming\ILP\

HKEY_LOCAL_MACHINE\SOFTWARE Wow6432Node**DOCMAESTRO\ILP**
Name: EXE Type: REG_SZ
Value: C:\{Program Files (x86)}\DocMaestro\ILP \

NOTE: On 32-bit machines the application path is **Program Files**\ instead of **Program Files (x86)**\ and the registry path does not include **Wow6432Node**\.

Challenge and Response

The ***InfoLinker Plus*** system uses a Challenge and Response process for license protection. The first time that ***InfoLinker Plus*** is used after installation, a dialog displays with a Challenge code and the PC name. The two sets of values should be sent to DocMaestro LLC for registration of the license. DocMaestro LLC returns a response code sequence that can be entered into the dialog. Once the correct response code is entered in the dialog the Challenge and Response Dialog will not appear again. The user can perform a number of operations prior to the entering of the correct response. After several uses without a correct response the ***InfoLinker Plus*** processing will stop working.

Support

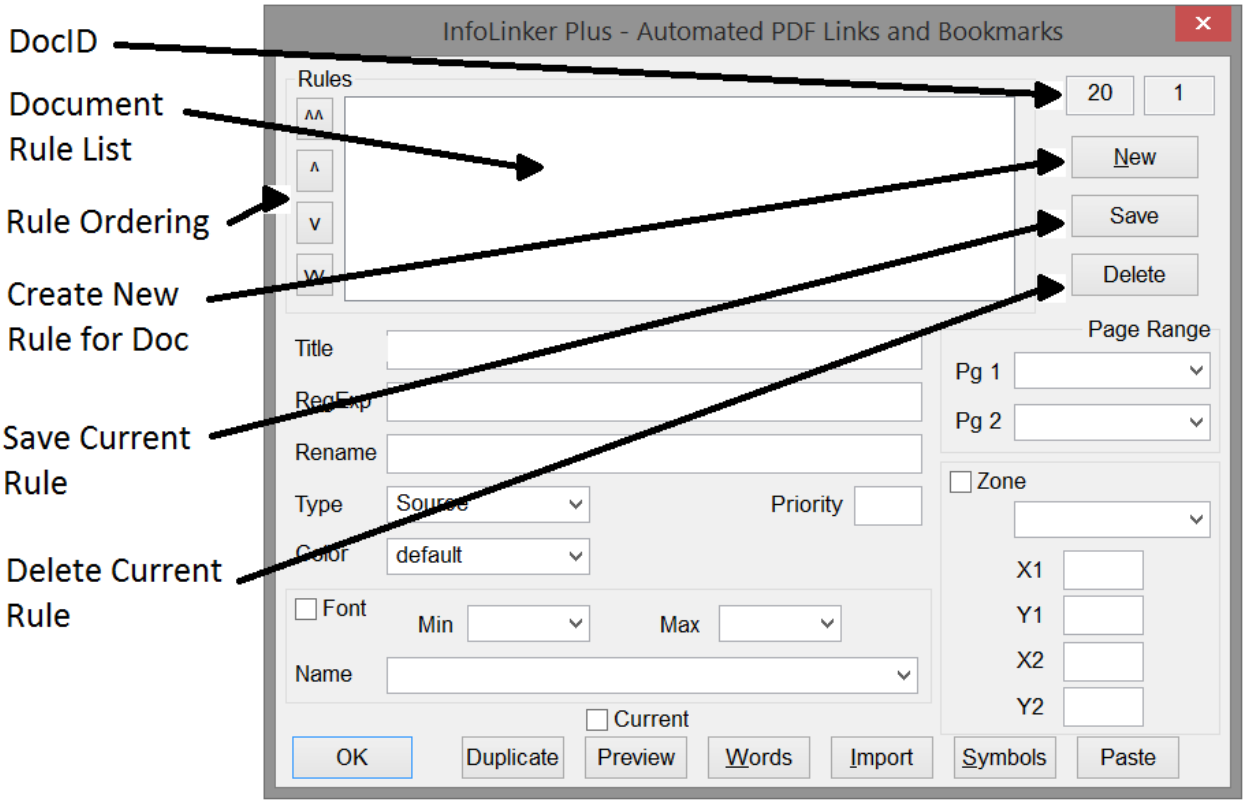
DocMaestro, LLC provides product support for the first year of each licensed product. This includes help with installation, problem resolutions, operational questions and support, operating system updates, and required Acrobat updates. On-going support renewal is available for purchase on a yearly basis starting at the anniversary date of installation of the product.

Chapter 2: Rule Writing Basics

The InfoLinker Plus Rules Editor

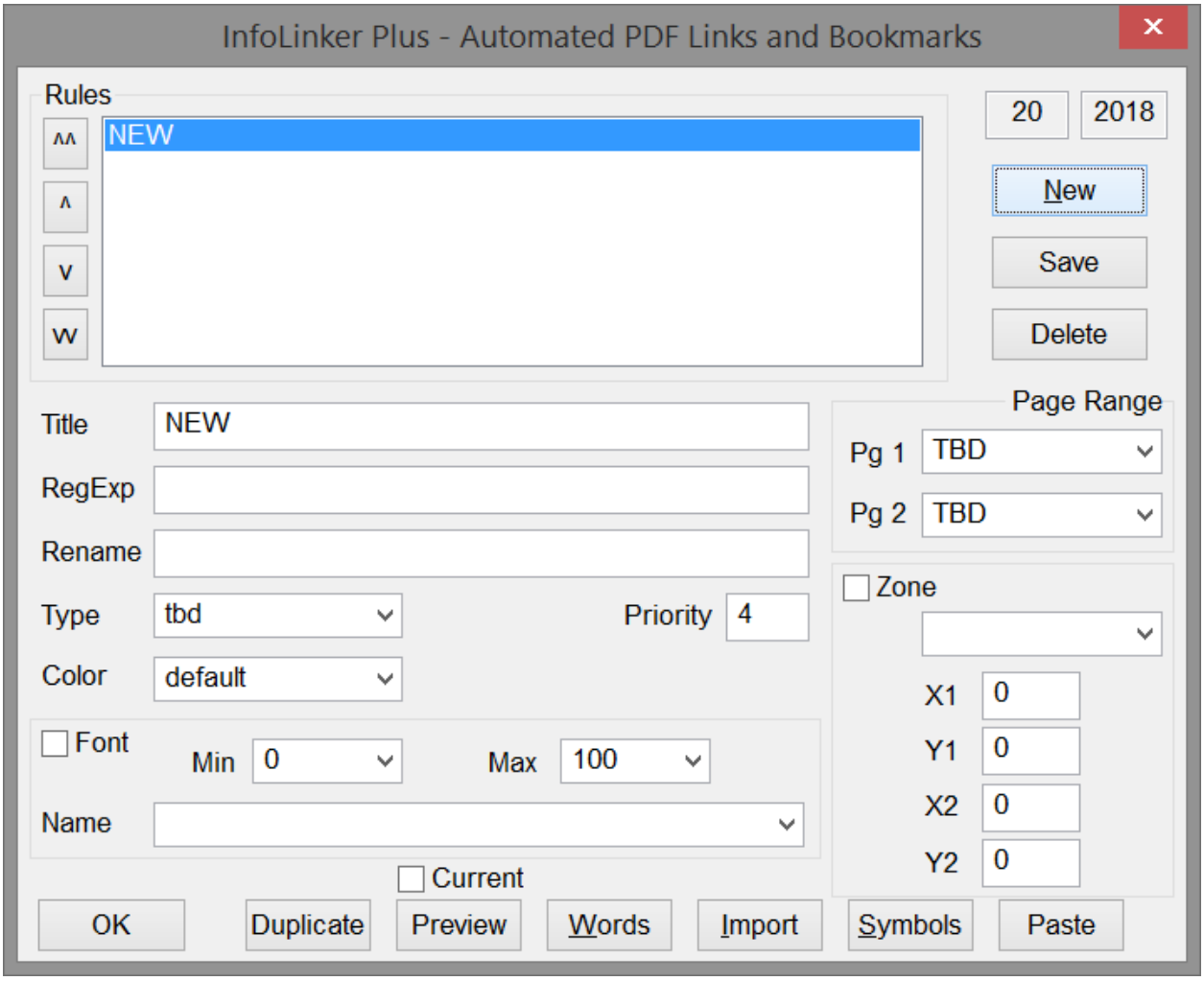
The first time that the ***InfoLinker Plus*** Rules Editor is opened for a PDF document the Rules Editor appears as below in figure 13. There are no rules listed in the Document Rule List. The top portion of this dialog is for managing the rules for the document. A rule writer can create "New" rules, "Delete" existing rules, change rule order, and "Save" an edited rule by using the appropriate buttons. Rule writers should remember to "Save" a rule if anything changes in the rule or the changes will be lost.

Figure 13. An Empty Rules Editor.



Adding a rule to the document is accomplished by pressing the "New" button. Figure 14 displays the result of pressing the "New" button the first time in a document.

Figure 14. A "New" rule in the Rules Editor.



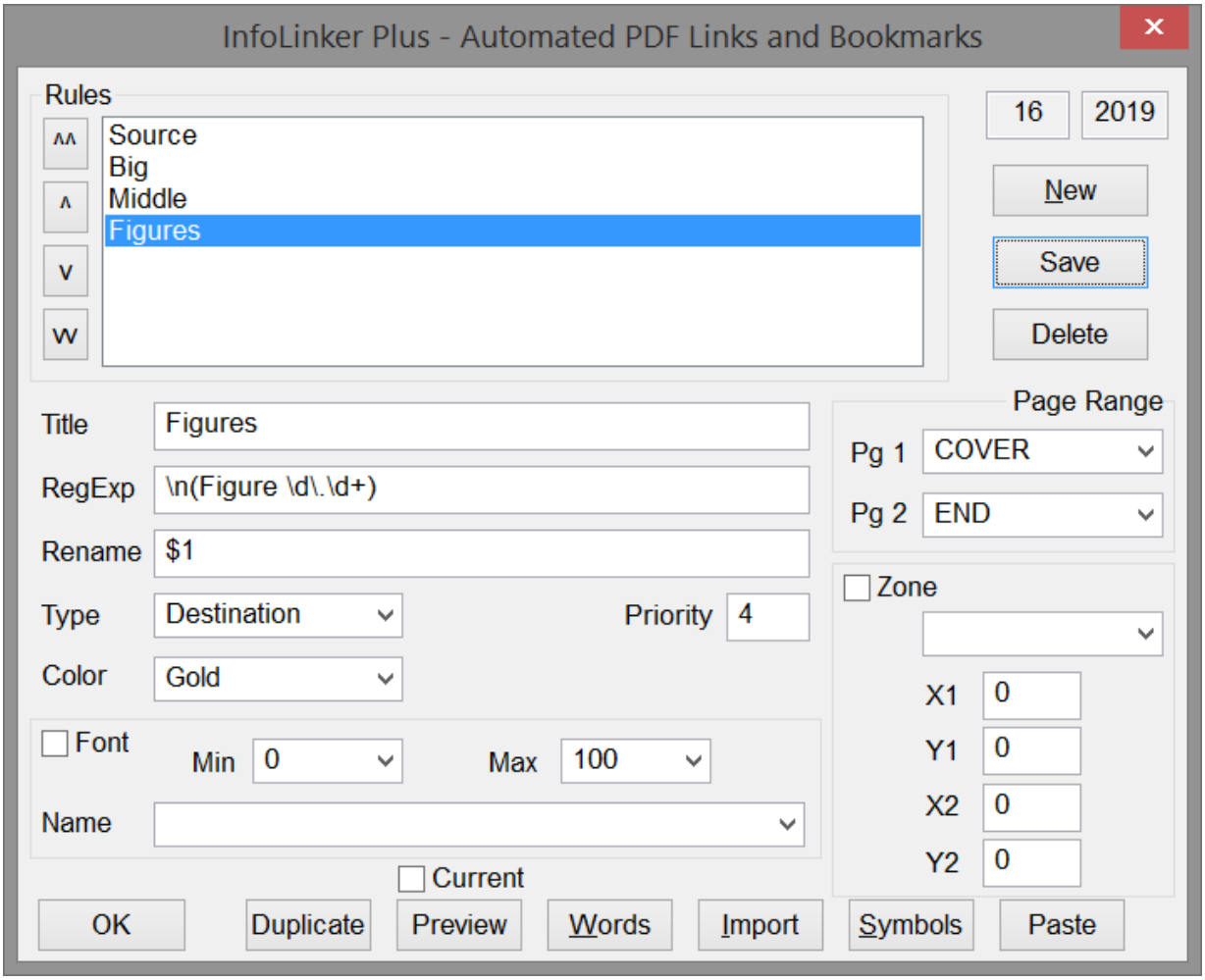
To edit this "New" rule the rule writer clicks on this new rule and its data fields are filled in to the appropriate fields on the dialog.

The fields below the document rule list box are used to define the currently selected rule. The "Title" field is the text that is displayed in the rule list box that represents this rule. The field can be any string of text the rule writer feels is appropriate. It is recommended that the "Title" field describe the rule in such a way that the rule writer can recognize what the rule is used for in the process. Since this is a "New" rule, the title fields as well as most of the other fields are blank. Some fields have a "TBD" in them. These fields will have to be modified before the "New" rule can be saved. A page range that the rule applies to must be entered. A "Rule Type" must be selected, and (in most cases) a "RegExp" must be written. The "Zone" section and the "Font" section are optional. They can be added to further qualify how a rule is applied for building links and bookmarks. The "Rename" field is used if a rule results in a string that needs to be adjusted in order to match a source or destination counterpart. The renaming may also be used to create modified strings for bookmarks or external links. An example would be the figures in a document are labeled "Figure 3." but references to the figures are "see fig. 3". The source

links ("see fig. 3") could use a rename from "fig." to "Figure" so that the results of the renamed source strings will match the destination strings. Matching of source and destination is required to create links.

Figure 15 displays a set of rules from a document. The "Figure" rule is selected from the rule list box and the fields for the "Figures" rules are displayed in the dialog. This "Figures" rule will result in Gold links. The results will be defined as potential "Destination" links. The actual text that is searched in the document by the rule is the RegExp of "\n(Figure \d\.\d+)". This RegExp looks for a carriage return followed by the string "FIGURE", followed by a space character and then a number (\d) followed by a period (\.) and another number with one or many digits (\d+). The parentheses are used to group the RegExp into parts.

Figure 15. A Simple Sample Rule.

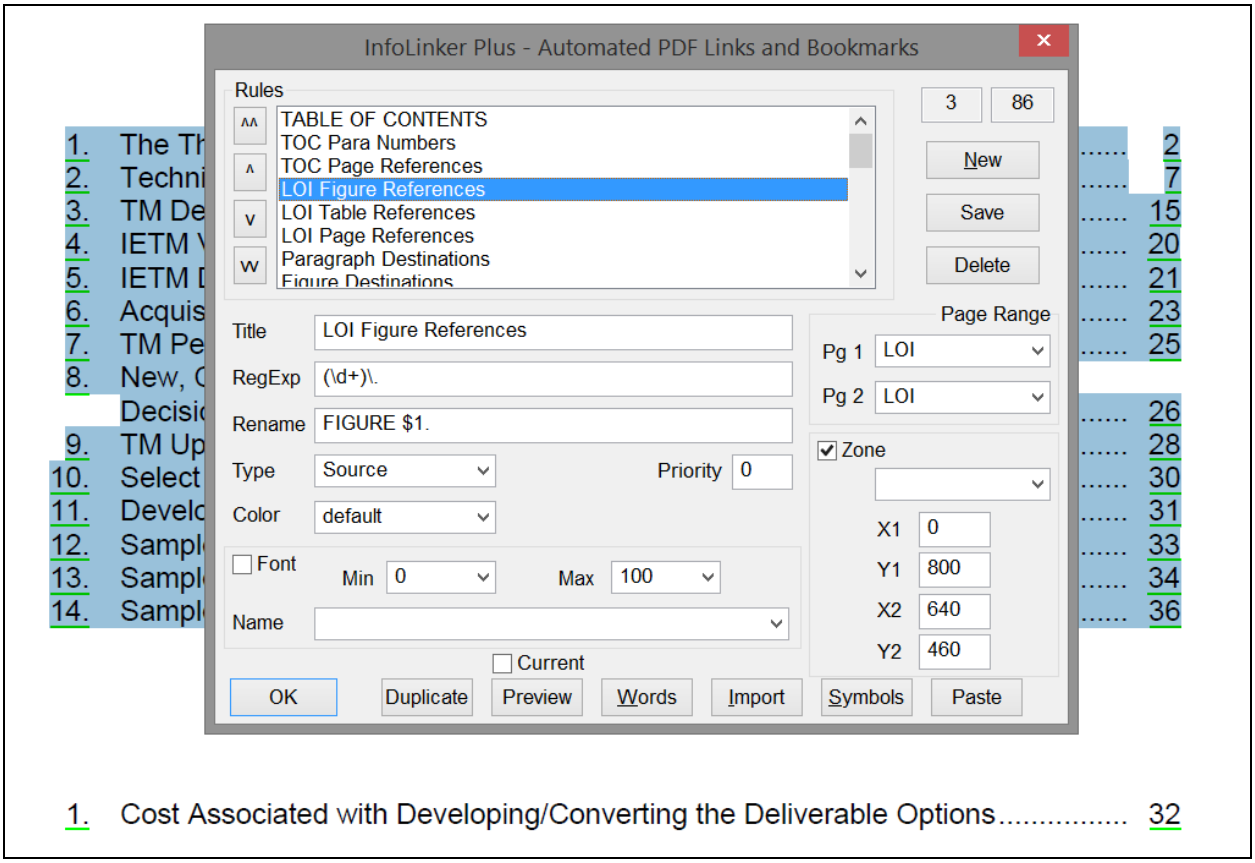


This "Figures" rule is used on all pages from "COVER" to "END". In InfoLinker Plus, COVER is always defined as PDF page 1 and END is the last page in the document. There is no "Font" or "Zone" data used by the rule. If the RegExp matches strings in the document within the page range then the rule will create bookmarks. Links to the results will only occur if there are source rule matches that exactly match the strings found by this rule.

Zone Definitions

Figure 16 displays a "LOI Figure References" rule. The displayed target page contains both the list of figures and list of tables. The rule shown was written to create links from the List of Figures which is on the top half of the page. A "Zone" is used to define whether the left hand numbers are figure or table numbers.

Figure 16. A Sample Zone Rule.



This rule uses the "Zone" to further clarify where on the page the rule should be applied. The target zone is the top-left of the page. Acrobat uses (0, 0) coordinate as the lower left corner of a page. In this rule X1 = 0 and X2 = 640 are the horizontal search values for matching on the page. This is the left part of the page. Y1 = 800 and Y2 = 460 are the vertical values that are used to restrict matching. The Y values increase up the page. So this zone is the top part of the page.

This "Figure TOC" rule only applies to the one page in this document. That page is the LOI page and is defined in the symbol table [not shown] as PDF page 8. This rule also uses the "Rename" field. The RegExp of the rule looks for numbers (\d+) followed by a period (\.) inside the zone. In order to match figure destinations in this document, the number in the potential link must be preceded by FIGURE and followed by a period. If the rule matched the text string "12." from the list of figures, it will be replaced by "FIGURE 12." as the result of the rename. If there is a destination result that is "FIGURE 12." Then the text in the zone that is "12." will have a link to "FIGURE 12." in the document. They must match to create a result.

To populate the zone fields, the rule writer can preselect the target text in the current PDF before opening the rules editor. Pressing the "Paste" button then fills in the Zone values for the selected text. "Paste" also places the selected text in the "RegExp" field as well as the related information in the "Font" field. To save this information the rule writer will need to select the check box of either the "Zone" section and/or the "Font" section of the rule before saving the rule. The above example was created by selecting the table of figures and then using the "Paste" button in the rules editor to paste the zone coordinates in the rule.

Zones can be named by the rule writer. Once the zone fields are complete a rule writer can edit the text in the text field under the "Zone" check box. Once the rule is saved then the zone values will also be saved with the name from the zone name field. The next time the rule writer needs to use the saved zone the drop down field of zone names will contain all the zone names that are saved with this document. By selected a name the zone field values will be recalled from memory and populate the zone fields for the current rule.

Link and Bookmark Color

There are default colors in ***InfoLinker Plus*** for creating source and destination links as well as bookmarks. Rules can override the default colors when they result in links and bookmarks. There are default values for source and destination links. The default bookmarks are black. The rule defined in figure 16 will result in default colors for source links. The rule in figure 15 has a color defined in the field next to the title. Gold is select from the collection of colors. Any destination links that are created from this rule will be gold and the bookmarks will also be gold.

Font Restrictions

The font of a text string match can be used to restrict a text string as a possible link or bookmark. The Rules Editor populates the font dropdown selector with all of the fonts in the current PDF document. When the dropdown is selected the list of fonts is displayed and a font can be selected for font restriction in the rule as shown in figure 17. The font restriction can be a specific font from the list or a regular expression (see chapter 3) can be used to select font characteristics. In figure 17 the string "Verdana" has been entered in the "Name" field. This will match any font that has "Verdana" in its name. From the list there are several fonts that contain the string "Verdana". They are:

"ABCDEE+Verdana,Italic",
"ABCDEE+Verdana,Bold",
"ABCDEE+Verdana,BoldItalic",
and "ABCDEE+Verdana".

Any matching text strings that are also made from one of these fonts will be a potential match for this rule. If a matching string is not one of thes fonts then the strings are eliminated as potential matches by ***InfoLinker Plus***. Another related example is that titles in most documents are usually of a Bold version of a font and using the font restriction of "Bold" could be very helpful in distinguishing between titles and standard document text.

Font size can also restrict string matching. The default values of Min=0 and Max=100 should allow any string to match. By changing these values a writer can restrict the number of potential matching strings. If the writer knows that only string with a font size greater that 14 points is required then setting Min=14 will restrict the number of matches by the rule.

A rule writer can select text in the PDF and then bring up the Rules Editor. The "Paste" button in the Rules Editor will populate the font "Name" field with the font name of the selected text in addition to the "Zone" of the selection. For the rule writer to determine font size they can use the "Words" button to display the "Words Dialog" for the current PDF page. Then look for the individual words on the page with their font name and size.

Figure 17. Font Selection in Rules Editor.

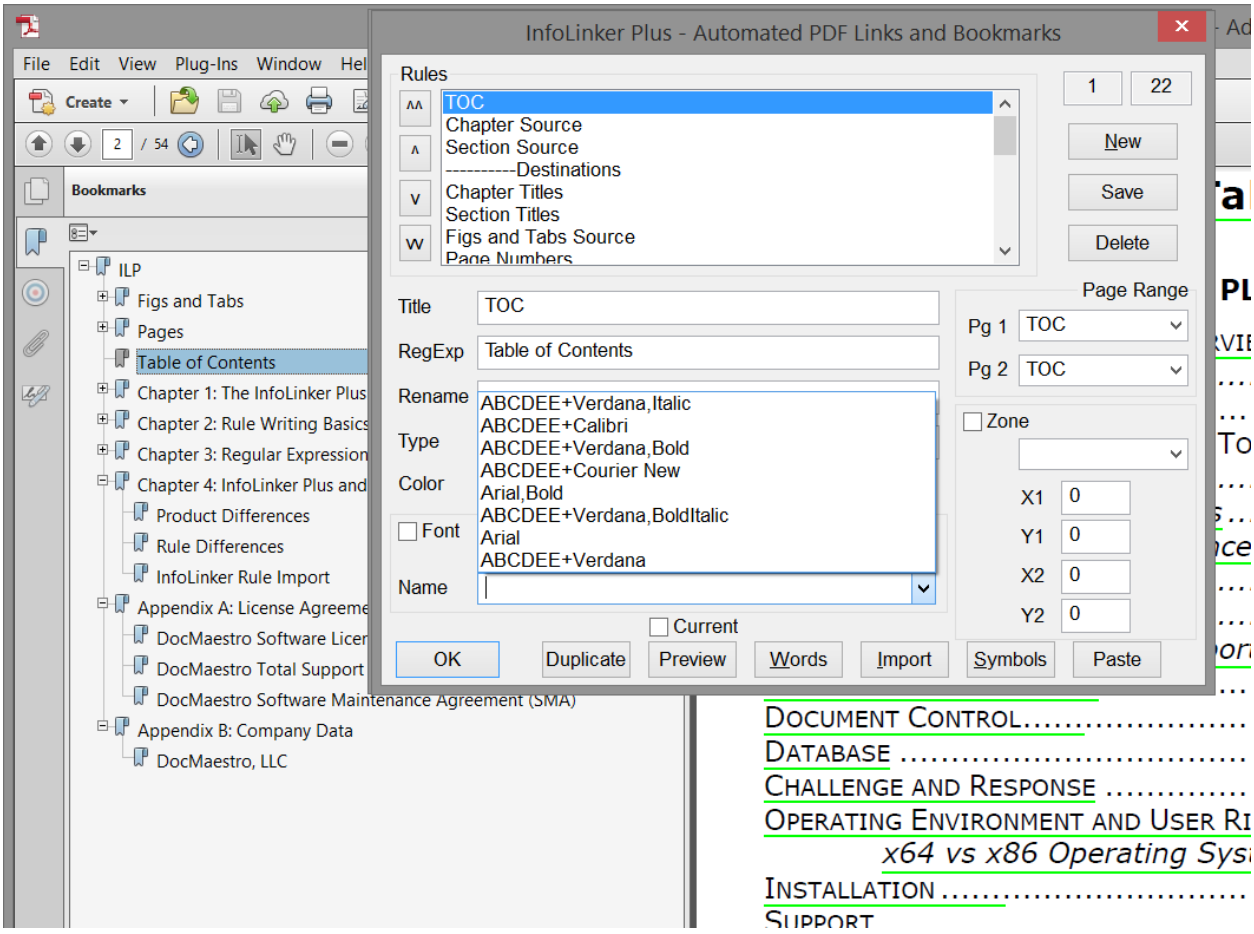
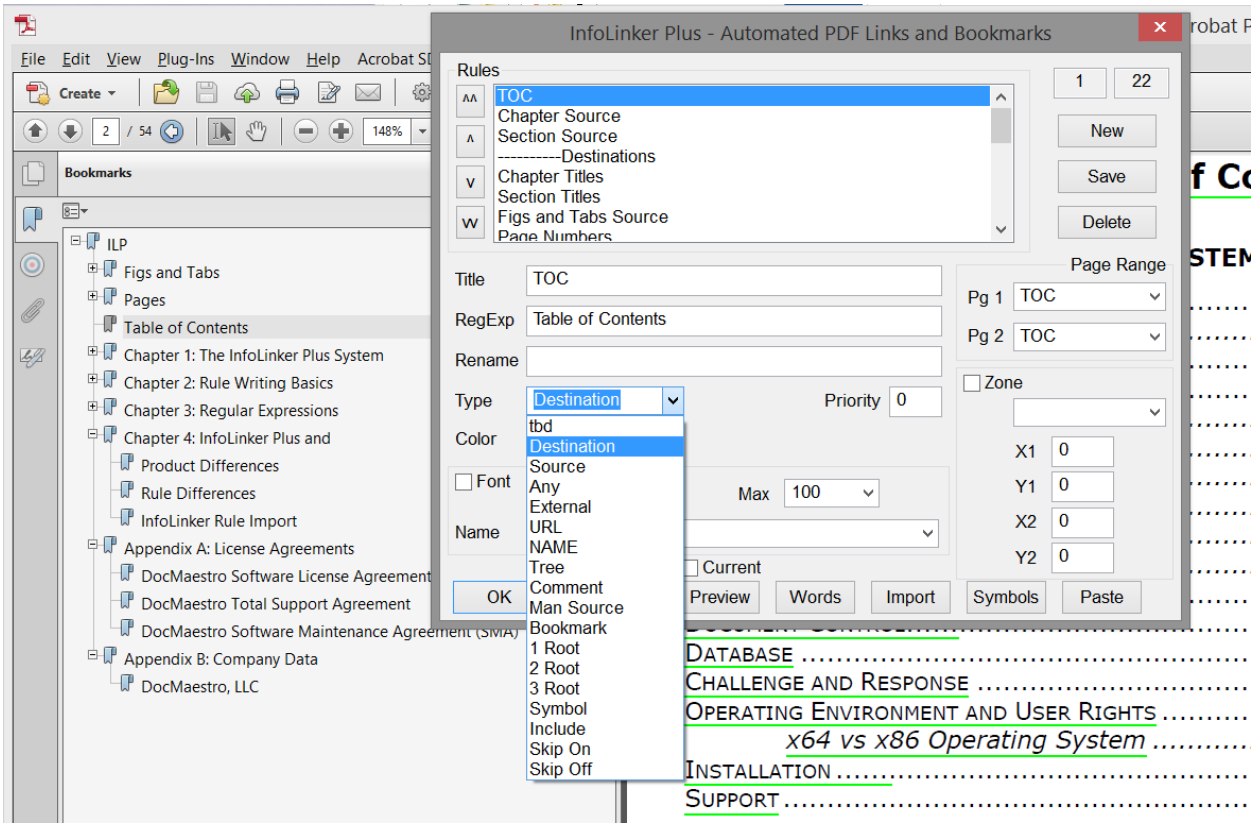


Figure 18. Rule Type Selection.



Rule Syntax

InfoLinker Plus uses standard Regular Expression syntax for building the RegExp and Rename of the rules. This Regular Expression Library is part of .Net Framework 4.0. The syntax is documented at: http://msdn.microsoft.com/en-us/library/hs600312.aspx. Most of the syntax used in InfoLinker can be used in the ***InfoLinker Plus*** rules. Some of the InfoLinker syntax is not Standard Regular Expression syntax, but ***InfoLinker Plus*** translates the InfoLinker specific syntax to the new standard when required. An example is the phrase break (%) used in the design of InfoLinker. This was done since many documents were scanned from paper and it was difficult to tell if the space between words was supposed to be a space character, tab character, line-feed character or even a carriage return. A phrase break represented all these values. Regular Expressions has a value that represents this set of values. It is "\s". But ***InfoLinker Plus*** also can distinguish a difference between each of the values represented by the phrase break so the rules will work better if the best representative value is used in the rule. A space is "\ ", a tab position is "\t", a line feed is "\l" and a carriage return is represented by "\n". These values are not available in InfoLinker rule development.

Some examples of other new values for RegExp in ***InfoLinker Plus*** include:

Table 2. Set of Regular Expression Values in *InfoLinker Plus* rules.

\w	Matches any word character.
\W	Matches any non-word character.
\s	Matches any white-space character.
\S	Matches any non-white-space character.
\d	Matches any decimal digit.
\D	Matches any character that is not a decimal digit.

Rule Basics

The basic approach that ***InfoLinker Plus*** is designed to do is build navigation aids in a PDF document. These aides are in the form of bookmarks and links. The basic example described earlier in this document consists of the two phrases "Figure 2.1" and the "see figure 2.1". The first phrase "Figure 2.1" is a thing that is a target of destination in the document. Readers may be interested in finding the specific figure in the document. We call that a destination. We want this type of thing in the document to be easy to find. We can make it a bookmark in a PDF and if it's organized in the bookmark tree then it will be easier for a reader to find, especially if it's listed between Figure 2.0 and Figure 2.2. Rules that locate destinations are called *Destination Rules.* The phrase "see figure 2.2" is related to the destination. Its letting the reader know that there is a reason to look at the figure that is somewhere in the document. This is what we call a Source. Rules that find sources in a PDF document are called *Source Rules*. There usually is at least one source rule for every destination rule in a set of rules for a PDF.

A rule writer can build a *Destination Rule* to find the "Figure 2.1" by writing a RegExp that is literally the text that is being search. In other words, the simplest RegExp for finding "Figure 2.1" is "Figure 2.1". This approach, although it works, can be verbose and difficult to build all of the rules. Also, if the document changes and "Figure 5.6" is added at the end of the document in a new version of the document, a new rule would need to be written for the new figure. Regular expressions allow a rule writer to build more generic RegExp strings to find all the occurrences of patterns in the document. The RegExp string "\d" will find a number character. Any single number digit can be found if we use "\d" in a rule. A period (".") is a special symbol. The period will match any typed character. In order to find a period the regular expression syntax requires that we use a literal character in combination with the period to be specific about finding the period (".") character in the text. "\." is the string used to find a period in a string of characters. Combining these search characters together we can find the pattern of "2.1" using "\d\.\d" in our rule. So the RegExp string for finding "Figure 2.1" can be made more general by using "Figure \d\.\d". This will find every figure that is in a document that includes "Figure 1.0" to "Figure 9.9".

Sometimes there are references to figures or sources that look a lot like the string that is the destination. So, sometimes we need to distinguish the source and the destination by recognizing other characteristics that differentiate the sources from the destination. Often the spacing around the destination is different from the source. Sometimes the font may be different, or the position or zone on the page differs between sources and destinations. Rule writing allows us to use any or all of these characteristics in the rules to make sure that the sources are different from destinations.

Rules can also vary based on the pages in the document. A Table of Figures in a document may list all of the figures, but the word "Figure" may not appear in the listing. In the above example, "Figure 2.1" may list the reference of source to the figure as "2.1". A specific rule can be applied to the Table of Figures that only uses the RegExp "\d\.\d" in a source rule that will build links to figure 2.1.

This brings up another part of rule writing. There is the ability to "rename" rules. The table of figures would have found "2.1", but there may be a paragraph labeled "2.1" in our document in addition to the "Figure 2.1". By adding a rename string to the rule we can distinguish the "2.1" from the table of figures from the paragraphs. The rename for the RegExp "\d\.\d" on the table of figures page would be "Figure $1". The $1 would be replaced by the first matching pattern in the RegExp of the rule. Parenthesis can be used to organize long RegExp into pieces that can each be used in the rename patterns. A RegExp like "(see|find) ((F|f)igure|(T|t)able) (\d\.\d)(\.|\,) will find:

Table 3. A RegExp to find Matching Strings.

RegExp
(see|find) ((F|f)igure|(T|t)able) (\d\.\d)(\.|\,)
Document Text
see figure 4.1.
find Figure 4.3,
see Table 2.8.
find table 1.1.

The rename portion of the rule can change what destinations the sources link to. A rename in the rule of "$2 $5" would result in the above sources becoming:

Table 4. A RegExp and Rename to find Matching Strings.

RegExp	Rename
(see|find) ((F|f)igure|(T|t)able) (\d\.\d)(\.|\,)	$2 $5
Document Text	Source Link
see figure 4.1.	figure 4.1
find Figure 4.3,	Figure 4.3
see Table 2.8.	Table 2.8
find table 1.1.	table 1.1

Notice that the $2 skips the first part of the string, the (see|find), and uses the second set of parenthesis. The third and fourth sets of parentheses are for finding either case of the first letter of either figure or table and the fifth set of parentheses is the number pattern that we want in the source rename. The parentheses are counted from left to right from where they begin.

A user may want to rename more of the source strings. There are many options. If we use a rename of "ITEM $5" it would result in:

Table 5. More Complicated Renames.

RegExp	Rename
(see|find) ((F|f)igure|(T|t)able) (\d\.\d)(\.|\,)	ITEM $5
Document Text	Source Link
see figure 4.1.	ITEM 4.1
find Figure 4.3,	ITEM 4.3
see Table 2.8.	ITEM 2.8
find table 1.1.	ITEM 1.1

There are a lot of options in how RegExp strings are written and the results of renaming items. Those options are covered in more detail in Chapter 3.

Chapter 3: Regular Expressions

InfoLinker Plus Uses Regular Expressions

InfoLinker Plus uses regular expressions for formulating the rules that are used to build links and bookmarks. This is a standard software methodology that has been used in the computing industry for text and pattern matching for many years. The remaining information in this chapter deals with regular expressions and it has been extracted from Wikipedia.

From Wikipedia, the free encyclopedia

Jump to: navigation, search

In computing, a **regular expression** provides a concise and flexible means for "matching" (specifying and recognizing) strings of text, such as particular characters, words, or patterns of characters. Abbreviations for "regular expression" include "regex" and "regexp". The concept of regular expressions was first popularized by utilities provided by Unix distributions, in particular the editor ed and the filter grep.[*citation needed*] A regular expression is written in a formal language that can be interpreted by a regular expression processor, which is a program that either serves as a parser generator or examines text and identifies parts that match the provided specification. Historically, the concept of regular expressions is associated with Kleene's formalism of regular sets, introduced in the 1950's.

Here are examples of specifications that could be expressed in a regular expression:

- the sequence of characters "car" appearing consecutively in any context, such as in "car", "cartoon", or "bicarbonate"
- the sequence of characters "car" occurring in that order with other characters between them, such as in "Icelander" or "chandler"
- the word "car" when it appears as an isolated word
- the word "car" when preceded by the word "blue" or "red"
- the word "car" when *not* preceded by the word "motor"
- a dollar sign immediately followed by one or more digits, and then optionally a period and exactly two more digits (for example, "$100" or "$245.99").

These examples are simple. Specifications of great complexity can be conveyed by regular expressions.

Regular expressions are used by many text editors, utilities, and programming languages to search and manipulate text based on patterns. Some of these languages, including Perl, Ruby, AWK, and Tcl, have been designed so that regular expressions are fully integrated into the syntax of the core language itself. Other programming languages like .NET languages, Java, and Python instead provide regular expressions through standard libraries. For yet other languages, such as Object Pascal, C and C++, non-core libraries are available (however, version C++11 provides regular expressions in its Standard Libraries).

As an example of the syntax, the regular expression `\bex` can be used to search for all instances of the string "*ex*" that occur after "word boundaries". Thus `\bex` will find the

matching string "ex" in two possible locations, (1) at the beginning of words, and (2) between two characters in a string, where the first is not a word character and the second is a word character. For instance, in the string "Texts for experts", `\bex` matches the "*ex*" in "experts" but not in "Texts" (because the "*ex*" occurs inside a word and not immediately after a word boundary).

Many modern computing systems provide wildcard characters in matching filenames from a file system. This is a core capability of many command-line shells and is also known as globbing. Wildcards differ from regular expressions in generally expressing only limited forms of patterns.

Basic concepts

A regular expression, often called a pattern, is an expression that specifies a set of strings. It is more concise to specify a set's members by rules (such as a pattern) than by a list. For example, the set containing the three strings "*Handel*", "*Händel*", and "*Haendel*" can be specified by the pattern `H(ä|ae?)ndel` (or alternatively, it is said that the pattern *matches* each of the three strings). In most formalisms, if there exists at least one regex that matches a particular set then there exist an infinite number of such expressions. Most formalisms provide the following operations to construct regular expressions.

Boolean "or"
: A vertical bar separates alternatives. For example, `gray|grey` can match "*gray*" or "*grey*".

Grouping
: Parentheses are used to define the scope and precedence of the operators (among other uses). For example, `gray|grey` and `gr(a|e)y` are equivalent patterns which both describe the set of "*gray*" and "*grey*".

Quantification
: A quantifier after a token (such as a character) or group specifies how often that preceding element is allowed to occur. The most common quantifiers are the question mark `?`, the asterisk `*` (derived from the Kleene star), and the plus sign `+` (Kleene cross).

- **?** The question mark indicates there is *zero or one* of the preceding element. For example, `colou?r` matches both "*color*" and "*colour*".
- ***** The asterisk indicates there is *zero or more* of the preceding element. For example, `ab*c` matches "*ac*", "*abc*", "*abbc*", "*abbbc*", and so on.
- **+** The plus sign indicates there is *one or more* of the preceding element. For example, `ab+c` matches "*abc*", "*abbc*", "*abbbc*", and so on, but not "*ac*".

These constructions can be combined to form arbitrarily complex expressions, much like one can construct arithmetical expressions from numbers and the operations **+**, **−**, **×**, and **÷**. For example, `H(ae?|ä)ndel` and `H(a|ae|ä)ndel` are both valid patterns which match the same strings as the earlier example, `H(ä|ae?)ndel`.

The precise syntax for regular expressions varies among tools and with context; more detail is given in the *Syntax* section.

History

Further information: Pattern matching#History

The origins of regular expressions lie in automata theory and formal language theory, both of which are part of theoretical computer science. These fields study models of computation (automata) and ways to describe and classify formal languages. In the 1950s, mathematician Stephen Cole Kleene described these models using his mathematical notation called *regular sets*.[1] The SNOBOL language was an early implementation of pattern matching, but not identical to regular expressions. Ken Thompson built Kleene's notation into the editor QED as a means to match patterns in text files. He later added this capability to the Unix editor ed, which eventually led to the popular search tool grep's use of regular expressions ("grep" is a word derived from the command for regular expression searching in the ed editor: `g/re/p` where *re* stands for regular expression[2]). Since that time, many variations of Thompson's original adaptation of regular expressions have been widely used in Unix and Unix-like utilities including expr, AWK, Emacs, vi, and lex.

Perl and Tcl regular expressions were derived from a regex library written by Henry Spencer, though Perl later expanded on Spencer's library to add many new features.[3] Philip Hazel developed PCRE (Perl Compatible Regular Expressions), which attempts to closely mimic Perl's regular expression functionality and is used by many modern tools including PHP and Apache HTTP Server. Part of the effort in the design of Perl 6 is to improve Perl's regular expression integration, and to increase their scope and capabilities to allow the definition of parsing expression grammars.[4] The result is a mini-language called Perl 6 rules, which are used to define Perl 6 grammar as well as provide a tool to programmers in the language. These rules maintain existing features of Perl 5.x regular expressions, but also allow BNF-style definition of a recursive descent parser via sub-rules.

The use of regular expressions in structured information standards for document and database modeling started in the 1960s and expanded in the 1980s when industry standards like ISO SGML (precursored by ANSI "GCA 101-1983") consolidated. The kernel of the structure specification language standards consists of regular expressions. Its use is evident in the DTD element group syntax.

Formal Language Theory

Definition

Regular expressions describe regular languages in formal language theory. They have thus the same expressive power as regular grammars. Regular expressions consist of constants and operators that denote sets of strings and operations over these sets, respectively. The following definition is standard, and found as such in most textbooks on formal language theory.[5][6] Given a finite alphabet Σ, the following constants are defined:

- (*empty set*) ∅ denoting the set ∅.
- (*empty string*) ε denoting the set containing only the "empty" string, which has no characters at all.
- (*literal character*) `a` in Σ denoting the set containing only the character *a*.

The following operations are defined:

- (*concatenation*) *RS* denoting the set { αβ | α in *R* and β in *S* }. For example {"ab", "c"}{"d", "ef"} = {"abd", "abef", "cd", "cef"}.
- (*alternation*) *R* | *S* denoting the set union of *R* and *S*. For example {"ab", "c"}|{"ab", "d", "ef"} = {"ab", "c", "d", "ef"}.
- (*Kleene star*) *R** denoting the smallest superset of *R* that contains ε and is closed under string concatenation. This is the set of all strings that can be made by concatenating any finite number (including zero) of strings from *R*. For example, {"0","1"}* is the set of all finite binary strings (including the empty string), and {"ab", "c"}* = {ε, "ab", "c", "abab", "abc", "cab", "cc", "ababab", "abcab", ... }.

To avoid parentheses it is assumed that the Kleene star has the highest priority, then concatenation and then set union. If there is no ambiguity then parentheses may be omitted. For example, `(ab)c` can be written as `abc`, and `a|(b(c*))` can be written as `a|bc*`. Many textbooks use the symbols ∪, +, or ∨ for alternation instead of the vertical bar.

Examples:

- `a|b*` denotes {ε, *a*, *b*, *bb*, *bbb*, ...}
- `(a|b)*` denotes the set of all strings with no symbols other than *a* and *b*, including the empty string: {ε, *a*, *b*, *aa*, *ab*, *ba*, *bb*, *aaa*, ...}
- `ab*(c|ε)` denotes the set of strings starting with *a*, then zero or more *b*s and finally optionally a *c*: {*a*, *ac*, *ab*, *abc*, *abb*, *abbc*, ...}

Expressive power and compactness

The formal definition of regular expressions is purposely parsimonious and avoids defining the redundant quantifiers `?` and `+`, which can be expressed as follows: `a+` = `aa*`, and `a?` = `(a|ε)`. Sometimes the complement operator is added, to give a *generalized regular expression*; here R^c matches all strings over Σ* that do not match *R*. In principle, the complement operator is redundant, as it can always be circumscribed by using the other operators. However, the process for computing such a representation is complex, and the result may require expressions of a size that is double exponentially larger.[7][8]

Regular expressions in this sense can express the regular languages, exactly the class of languages accepted by deterministic finite automata. There is, however, a significant difference in compactness. Some classes of regular languages can only be described by deterministic finite automata whose size grows exponentially in the size of the shortest equivalent regular expressions. The standard example are here the languages L_k consisting of all strings over the alphabet {*a*,*b*} whose k^{th}-last letter equals *a*. On the one hand, a regular expression describing L_4 is given by $(a \mid b)^* a(a \mid b)(a \mid b)(a \mid b)$. Generalizing this pattern to L_k gives the expression

$$(a|b)^* a \underbrace{(a|b)(a|b)\cdots(a|b)}_{k-1 \text{ times}}.$$

On the other hand, it is known that every deterministic finite automaton accepting the language L_k must have at least 2^k states. Luckily, there is a simple mapping from regular expressions to the more general nondeterministic finite automata (NFAs) that does not lead to such a blowup in size; for this reason NFAs are often used as alternative representations of regular languages. NFAs are a simple variation of the type-3 grammars of the Chomsky hierarchy.[5]

Finally, it is worth noting that many real-world "regular expression" engines implement features that cannot be described by the regular expressions in the sense of formal language theory; see below for more on this.

Deciding equivalence of regular expressions

As seen in many of the examples above, there is more than one way to construct a regular expression to achieve the same results.

It is possible to write an algorithm which for two given regular expressions decides whether the described languages are essentially equal, reduces each expression to a minimal deterministic finite state machine, and determines whether they are isomorphic (equivalent).

The redundancy can be eliminated by using Kleene star and set union to find an interesting subset of regular expressions that is still fully expressive, but perhaps their use can be restricted. This is a surprisingly difficult problem. As simple as the regular expressions are, there is no method to systematically rewrite them to some normal form. The lack of axiom in the past led to the star height problem. Recently, Dexter Kozen axiomatized regular expressions with Kleene algebra.[9]

Syntax

A number of special characters or meta characters are used to denote actions or delimit groups; but it is possible to force these special characters to be interpreted as normal characters by preceding them with a defined escape character, usually the backslash "\". For example, a dot is normally used as a "wild card" metacharacter to denote any character, but if preceded by a backslash it represents the dot character itself. The pattern `c.t` matches "cat", "cot", "cut", and non-words such as "czt" and "c.t"; but `c\.t` matches only "c.t". The backslash also escapes itself, i.e., two backslashes are interpreted as a literal backslash character.

POSIX

POSIX Basic Regular Expressions

Traditional Unix regular expression syntax followed common conventions but often differed from tool to tool. The IEEE POSIX Basic Regular Expressions (BRE) standard (released alongside an alternative flavor called Extended Regular Expressions or ERE) was designed mostly for backward compatibility with the traditional (Simple Regular Expression) syntax but provided a common standard which has since been adopted as the default syntax of many Unix regular expression tools, though there is often some variation or additional features. Many such tools also provide support for ERE syntax with command line arguments.

In the BRE syntax, most characters are treated as literals — they match only themselves (e.g., `a` matches "*a*"). The exceptions, listed below, are called metacharacters or metasequences.

Metacharacter	Description
`.`	Matches any single character (many applications exclude newlines, and exactly which characters are considered newlines is flavor, character encoding, and platform specific, but it is safe to assume that the line feed character is included). Within POSIX bracket expressions, the dot character matches a literal dot. For example, `a.c` matches "*abc*", etc., but `[a.c]` matches only "*a*", "*.*", or "*c*".
`[ ]`	A bracket expression. Matches a single character that is contained within the brackets. For example, `[abc]` matches "*a*", "*b*", or "*c*". `[a-z]` specifies a range which matches any lowercase letter from "*a*" to "*z*". These forms can be mixed: `[abcx-z]` matches "*a*", "*b*", "*c*", "*x*", "*y*", or "*z*", as does `[a-cx-z]`. The `-` character is treated as a literal character if it is the last or the first (after the `^`) character within the brackets: `[abc-]`, `[-abc]`. Note that backslash escapes are not allowed. The `]` character can be included in a bracket expression if it is the first (after the `^`) character: `[]abc]`.
`[^ ]`	Matches a single character that is not contained within the brackets. For example, `[^abc]` matches any character other than "*a*", "*b*", or "*c*". `[^a-z]` matches any single character that is not a lowercase letter from "*a*" to "*z*". As above, literal characters and ranges can be mixed.
`^`	Matches the starting position within the string. In line-based tools, it matches the starting position of any line.
`$`	Matches the ending position of the string or the position just before a string-ending newline. In line-based tools, it matches the ending position of any line.
BRE: `\( \)` ERE: `( )`	Defines a marked subexpression. The string matched within the parentheses can be recalled later (see the next entry, `\n`). A marked subexpression is also called a block or capturing group.
`\n`	Matches what the *n*th marked subexpression matched, where *n* is a digit from 1 to 9. This construct is theoretically **irregular** and was not adopted in the POSIX ERE syntax. Some tools allow referencing more than nine capturing groups.
`*`	Matches the preceding element zero or more times. For example, `ab*c` matches "*ac*", "*abc*", "*abbbc*", etc. `[xyz]*` matches "", "*x*", "*y*", "*z*", "*zx*", "*zyx*", "*xyzzy*", and so on. `\(ab\)*` matches "", "*ab*", "*abab*", "*ababab*", and so on.
BRE: `\{m,n\}` ERE: `{m,n}`	Matches the preceding element at least *m* and not more than *n* times. For example, `a\{3,5\}` matches only "*aaa*", "*aaaa*", and "*aaaaa*". This is not found in a few older instances of regular expressions.

Examples:

- `.at` matches any three-character string ending with "at", including "*hat*", "*cat*", and "*bat*".
- `[hc]at` matches "*hat*" and "*cat*".
- `[^b]at` matches all strings matched by `.at` except "*bat*".
- `^[hc]at` matches "*hat*" and "*cat*", but only at the beginning of the string or line.
- `[hc]at$` matches "*hat*" and "*cat*", but only at the end of the string or line.
- `\[.\]` matches any single character surrounded by "[" and "]" since the brackets are escaped, for example: "*[a]*" and "*[b]*".

POSIX Extended Regular Expressions

The meaning of metacharacters escaped with a backslash is reversed for some characters in the POSIX Extended Regular Expression (ERE) syntax. With this syntax, a backslash causes the metacharacter to be treated as a literal character. So, for example, `\( \)` is now `( )` and `\{ \}` is now `{ }`. Additionally, support is removed for `\n` backreferences and the following metacharacters are added:

Metacharacter	Description
`?`	Matches the preceding element zero or one time. For example, `ba?` matches "*b*" or "*ba*".
`+`	Matches the preceding element one or more times. For example, `ba+` matches "*ba*", "*baa*", "*baaa*", and so on.
`|`	The choice (aka alternation or set union) operator matches either the expression before or the expression after the operator. For example, `abc|def` matches "*abc*" or "*def*".

Examples:

- `[hc]+at` matches "*hat*", "*cat*", "*hhat*", "*chat*", "*hcat*", "*ccchat*", and so on, but not "*at*".
- `[hc]?at` matches "*hat*", "*cat*", and "*at*".
- `[hc]*at` matches "*hat*", "*cat*", "*hhat*", "*chat*", "*hcat*", "*ccchat*", "*at*", and so on.
- `cat|dog` matches "*cat*" or "*dog*".

POSIX Extended Regular Expressions can often be used with modern Unix utilities by including the command line flag *-E*.

POSIX character classes

Since many ranges of characters depend on the chosen locale setting (i.e., in some settings letters are organized as *abc...zABC...Z*, while in some others as *aAbBcC...zZ*), the POSIX standard defines some classes or categories of characters as shown in the following table:

POSIX	Non-standard	Perl	ASCII	Description
[:alnum:]			[A-Za-z0-9]	Alphanumeric characters
	[:word:]	\w	[A-Za-z0-9_]	Alphanumeric characters plus "_"
		\W	[^A-Za-z0-9_]	Non-word characters
[:alpha:]			[A-Za-z]	Alphabetic characters
[:blank:]			[\t]	Space and tab
		\b	[(?<=\W)(?=\w)|(?<=\w)(?=\W)]	Word boundaries
[:cntrl:]			[\x00-\x1F\x7F]	Control characters
[:digit:]		\d	[0-9]	Digits
		\D	[^0-9]	Non-digits
[:graph:]			[\x21-\x7E]	Visible characters
[:lower:]			[a-z]	Lowercase letters
[:print:]			[\x20-\x7E]	Visible characters and the space character
[:punct:]			[\]\[!"#$%&'()*+,./:;<=>?@\^_`{|}~-]	Punctuation characters
[:space:]		\s	[\t\r\n\v\f]	Whitespace characters
		\S	[^ \t\r\n\v\f]	Non-whitespace characters
[:upper:]			[A-Z]	Uppercase letters
[:xdigit:]			[A-Fa-f0-9]	Hexadecimal digits

POSIX character classes can only be used within bracket expressions. For example, [[:upper:]ab] matches the uppercase letters and lowercase "*a*" and "*b*".

An additional non-POSIX class understood by some tools is [:word:], which is usually defined as [:alnum:] plus underscore. This reflects the fact that in many programming languages these are the characters that may be used in identifiers. The editor Vim further distinguishes *word* and *word-head* classes (using the notation \w and \h) since in many programming languages the characters that can begin an identifier are not the same as those that can occur in other positions.

Note that what the POSIX regular expression standards call *character classes* are commonly referred to as *POSIX character classes* in other regular expression flavors which support them. With most other regular expression flavors, the term *character class* is used to describe what POSIX calls *bracket expressions*.

Perl-derived regular expressions

Perl has a more consistent and richer syntax than the POSIX basic (BRE) and extended (ERE) regular expression standards. An example of its consistency is that `\` always escapes a non-alphanumeric character. Other examples of functionality possible with Perl but not POSIX-compliant regular expressions is the concept of lazy quantification (see the next section), possessive quantifies to control backtracking, named capture groups, and recursive patterns.

Due largely to its expressive power, many other utilities and programming languages have adopted syntax similar to Perl's — for example, Java, JavaScript, PCRE, Python, Ruby, Microsoft's .NET Framework, and the W3C's XML Schema all use regular expression syntax similar to Perl's. Some languages and tools such as Boost and PHP support multiple regular expression flavors. Perl-derivative regular expression implementations are not identical, and all implement no more than a subset of Perl's features, usually those of Perl 5.0, released in 1994. With Perl 5.10, this process has come full circle with Perl incorporating syntactic extensions originally developed in Python, PCRE, and the .NET Framework[*citation needed*]

Simple Regular Expressions

Simple Regular Expressions is a syntax that may be used by historical versions of application programs, and may be supported within some applications for the purpose of providing backward compatibility. It is deprecated.[10]

Lazy quantification

The standard quantifiers in regular expressions are greedy, meaning they match as much as they can. For example, to find the first instance of an item between the angled bracket symbols < > in this example:

```
Another whale sighting occurred on <January 26>, <2004>.
```

Someone new to regexes would likely come up with the pattern `<.*>` or similar. However, instead of the "*<January 26>*" that might be expected, this pattern will actually return "*<January 26>, <2011>*" because the `*` quantifier is greedy - it will consume as many characters as possible from the input, and "*January 26>, <2011*" has more characters than "*January 26*".

Though this problem can be avoided in a number of ways (e.g., by specifying the text that is *not* to be matched: `<[^>]*>`), modern regular expression tools allow a quantifier to be specified as *lazy* (also known as *non-greedy*, *reluctant*, *minimal*, or *ungreedy*) by putting a question mark after the quantifier (e.g., `<.*?>`), or by using a modifier which reverses the greediness of quantifiers (though changing the meaning of the standard quantifiers can be confusing). By using a lazy quantifier, the expression tries the minimal match first. Though in the previous example lazy matching is used to select one of many matching results, in some cases it can also be used to improve performance when greedy matching would require more backtracking.

Uses

Regular expressions are useful in the production of syntax highlighting systems, data validation, and many other tasks.

While regular expressions would be useful on search engines such as Google, processing them across the entire database could consume excessive computer resources depending on the complexity and design of the regex. Although in many cases system administrators can run regex-based queries internally, most search engines do not offer regex support to the public. Notable exceptions: Google Code Search, Exalead.

Conventions

The following conventions are used in the examples.[1]

```
metacharacter(s) ;; the metacharacters column specifies the regex syntax
                            being demonstrated
=~ m//           ;; indicates a regex match operation in Perl
=~ s///          ;; indicates a regex substitution operation in Perl
```

Also worth noting is that these regular expressions are all Perl-like syntax. Standard POSIX regular expressions are different.

Examples

Unless otherwise indicated, the following examples conform to the Perl programming language, release 5.8.8, January 31, 2006. This means that other implementations may lack support for some parts of the syntax shown here (e.g. basic vs. extended regex, \(\) vs. (), or lack of \d instead of POSIX [:digit:]).

The syntax and conventions used in these examples coincide with that of other programming environments as well (e.g., see *Java in a Nutshell* — Page 213, *Python Scripting for Computational Science* — Page 320, Programming PHP — Page 106).

Metacharacter(s)	Description	Example Note that all the if statements return a TRUE value

.	Normally matches any character except a newline. Within square brackets the dot is literal.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/...../) {`
		`  print "$string1 has length >= 5\n";`
		`}`

( )	Groups a series of pattern elements to a single element. When you match a pattern within parentheses, you can use any of $1, $2, ... later to refer to the previously matched pattern.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/(H..).(o..)/) {`
		`  print "We matched '$1' and '$2'\n";`
		`}`
		Output:
		`We matched 'Hel' and 'o W';`

+	Matches the preceding pattern element one or more times.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/l+/) {`
		`  print "There are one or more consecutive letter \"l\"'s in $string1\n";`
		`}`
		Output:
		`There are one or more consecutive letter "l"'s in Hello World`

?	Matches the preceding pattern element zero or one times.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/H.?e/) {`
		`  print "There is an 'H' and a 'e' separated by ";`
		`  print "0-1 characters (Ex: He Hoe)\n";`
		`}`

?	Modifies the *, +, or {M,N}'d regex that comes before to match as few times as possible.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/(l.+?o)/) {`
		`  print "The non-greedy match with 'l' followed by one or ";`
		`  print "more characters is 'llo' rather than 'llo wo'.\n";`
		`}`

*	Matches the preceding pattern element zero or more times.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/el*o/) {`
		`  print "There is an 'e' followed by zero to many ";`
		`  print "'l' followed by 'o' (eo, elo, ello, elllo)\n";`
		`}`

{M,N}	Denotes the minimum M and the maximum N match count.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/l{1,2}/) {`
		` print "There exists a substring with at least 1 ";`
		` print "and at most 2 l's in $string1\n";`
		`}`

[...]	Denotes a set of possible character matches.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/[aeiou]+/) {`
		`  print "$string1 contains one or more vowels.\n";`
		`}`

Metacharacter(s)	Description	Example

|	Separates alternate possibilities.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/(Hello|Hi|Pogo)/) {`
		`  print "At least one of Hello, Hi, or Pogo is ";`
		`  print "contained in $string1.\n";`
		`}`

\b	Matches a zero-width boundary between a word-class character (see next) and either a non-word class character or an edge.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/llo\b/) {`
		`  print "There is a word that ends with 'llo'\n";`
		`}`

\w	Matches an alphanumeric character, including "_"; same as [A-Za-z0-9_] in ASCII. In Unicode[2] same as [\p{Alphabetic}\p{GC=Mark}\p{GC=Decimal_Number\p{GC=Connector_Punctuation}], where the Alphabetic property contains more than just Letters, and the Decimal_Number property contains more than [0-9].	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/\w/) {`
		`  print "There is at least one alphanumeric";`
		`  print "character in $string1 (A-Z, a-z, 0-9, _)\n";`
		`}`

\W	Matches a **non**-alphanumeric character, excluding "_"; same as [^A-Za-z0-9_] in ASCII, and [^\p{Alphabetic}\p{GC=Mark}\p{GC=Decimal_Number}\p{GC=Connector_Punctuation}] in Unicode.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/\W/) {`
		`  print "The space between Hello and ";`
		`  print "World is not alphanumeric\n";`
		`}`

\s	Matches a whitespace character, which in ASCII are tab, line feed, form feed, carriage return, and space; in Unicode, also matches no-break spaces, next line, and the variable-width spaces (amongst others).	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/\s.*\s/) {`
		`  print "There are TWO whitespace characters, which may";`
		`  print " be separated by other characters, in $string1";`
		`}`

\S	Matches anything BUT a whitespace.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/\S.*\S/) {`
		`  print "There are TWO non-whitespace characters, which";`
		`  print " may be separated by other characters, in $string1";`
		`}`

\d	Matches a digit; same as [0-9] in ASCII; in Unicode, same as the \p{Digit} or \p{GC=Decimal_Number} property, which itself the same as the \p{Numeric_Type=Decimal} property.	`$string1 = "99 bottles of beer on the wall.";`
		`if ($string1 =~ m/(\d+)/) {`
		`  print "$1 is the first number in '$string1'\n";`
		`}`
		Output:
		`99 is the first number in '99 bottles of beer on the wall.'`

Metacharacter(s)	Description	Example

\D	Matches a non-digit; same as [^0-9] in ASCII or \P{Digit} in Unicode.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/\D/) {`
		`  print "There is at least one character in $string1";`
		`  print " that is not a digit.\n";`
		`}`

^	Matches the beginning of a line or string.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/^He/) {`
		`  print "$string1 starts with the characters 'He'\n";`
		`}`

$	Matches the end of a line or string.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/rld$/) {`
		`  print "$string1 is a line or string ";`
		`  print "that ends with 'rld'\n";`
		`}`

\A	Matches the beginning of a string (but not an internal line).	`$string1 = "Hello\nWorld\n";`
		`if ($string1 =~ m/\AH/) {`
		`  print "$string1 is a string ";`
		`  print "that starts with 'H'\n";`
		`}`

\z	Matches the end of a string (but not an internal line).
	see Perl Best Practices — Page 240
		`$string1 = "Hello\nWorld\n";`
		`if ($string1 =~ m/d\n\z/) {`
		`  print "$string1 is a string ";`
		`  print "that ends with 'd\\n'\n";`
		`}`

[^...]	Matches every character except the ones inside brackets.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/[^abc]/) {`
		`  print "$string1 contains a character other than ";`
		`  print "a, b, and c\n";`
		`}`

Chapter 4: InfoLinker Plus and InfoLinker

Product Differences

InfoLinker was first released in 1993. Acrobat was in version 1. The Borland Paradox Database Engine was used for data management. Windows current version was Windows 3.0. InfoLinker used a proprietary regular expression engine, compiler and builder that were developed by Alliant Tech Systems in the late 1980's for another document management product called AnyMedia. The text strings from PDFs were extracted from the PDF format and placed in an XML file structure that could be read by the regular expression system. The files were called OCR files and they were a rough approximation of the text positions from the PDF. There was no font information other than an approximate height.

InfoLinker has been periodically revised and upgraded to stay current with new releases by Adobe through Acrobat 9.0. The recent Adobe Acrobat X (10.0) and XI (11.0) release and the move to 64-bit architecture for PC platforms signaled an end to the life cycle of the InfoLinker product. A totally new product, built on the knowledge and experience of InfoLinker was developed.

InfoLinker Plus was initially designed to work in Windows 7. Development and testing used Acrobat 9 and Acrobat X. ***InfoLinker Plus*** is integrated with Microsoft Access 2010. The position and font of each word in the PDF is used by the rule processing. The regular expression engine used in ***InfoLinker Plus*** is the Microsoft .Net Regular Expression Library from version 2010.

Rule Differences

The rules in InfoLinker are text script files that are edited with a text editor. InfoLinker used directives to define where and how rules were applied to the OCR files generated by InfoLinker. Directives include; #PAGE, #PAGES, #ZONE, #SOURCE, #DESTINATION, #EITHER, #NAME, #SOURCE EXECUTABLE, #CMDLINE, #WORKDIR, and #INCLUDE. What lines are not directives or comments are rules written as regular expressions. There is a rule renaming syntax that used "<<" after the rule and the rename patterns use brackets and the number of the source rule pattern.

Since 2002 the regular expression standards have evolved. ***InfoLinker Plus*** attempts to follow these standards as best it can. The renaming also follows the evolved standards. The following table illustrated some of the changes in the RegExp patterns.

Table 6. InfoLinker Plus and InfoLinker Differences.

Definition	**InfoLinker**	***InfoLinker Plus***
Digits	:d	\d
Alpha characters	:a	\a
Alphanumeric (word character)	:n	\w
Non word character	*NA*	\W
Lowercase	:l	[a-z]
Uppercase	:u	[A-Z]
Rename patterns	[1], [n]	$1, $n
Phase break (unique to IL)	%	*NA*
White space character	*NA*	\s
Non-white space character	*NA*	\S
Beginning of page	^	^
End of page	$	$
New line	*NA*	\n
Tab	*NA*	\t

NA - not available

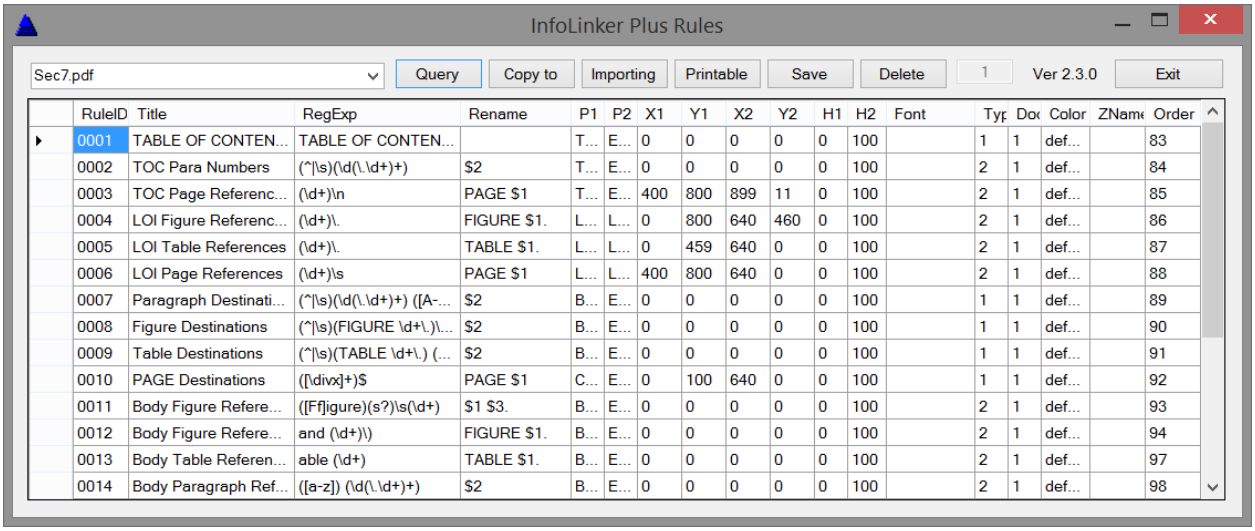
InfoLinker Rule Import

InfoLinker Plus ships with a program called RegRules.
The program is located as:

Program Files (x86)\DocMaestro\ILP\RegRules.exe

The field in the upper left corner of RegRules is the DocID of the current document displayed. If the number is 0 then all the rules in the database will be displayed after pressing the "Query" button.

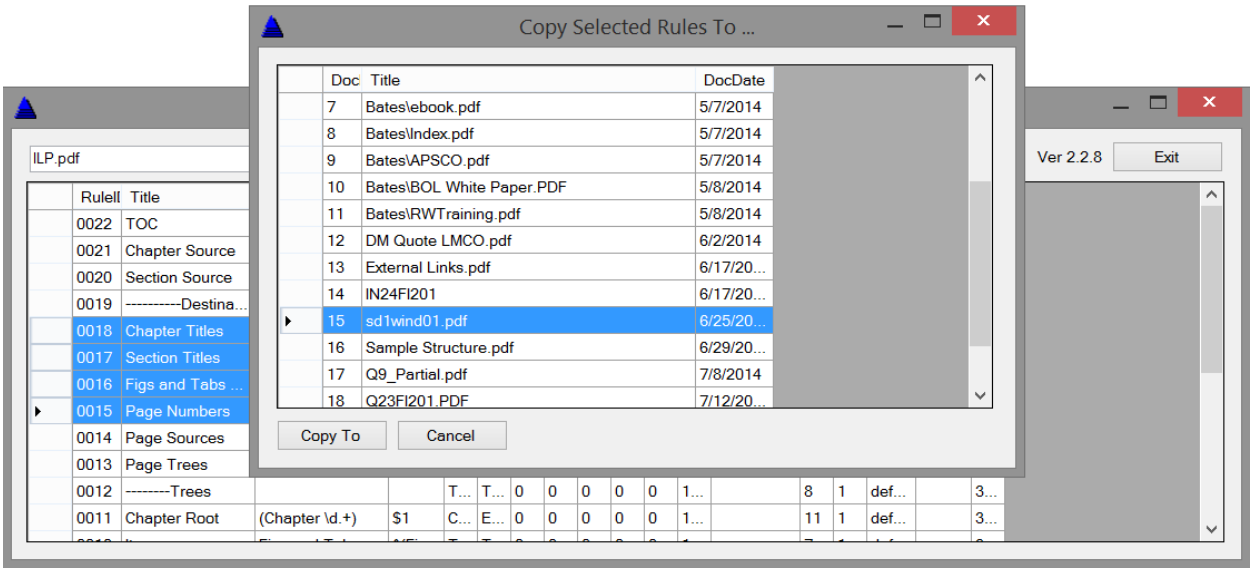
Figure 19. InfoLinker Plus Rules Program.



RuleID	Title	RegExp	Rename	P1	P2	X1	Y1	X2	Y2	H1	H2	Font	Typ	Doc	Color	ZName	Order
0001	TABLE OF CONTEN...	TABLE OF CONTEN...		T...	E...	0	0	0	0	0	100		1	1	def...		83
0002	TOC Para Numbers	(^\s)(\d(\.\d+)+)	$2	T...	E...	0	0	0	0	0	100		2	1	def...		84
0003	TOC Page Referenc...	(\d+)\n	PAGE $1	T...	E...	400	800	899	11	0	100		2	1	def...		85
0004	LOI Figure Referenc...	(\d+)\.	FIGURE $1.	L...	L...	0	800	640	460	0	100		2	1	def...		86
0005	LOI Table References	(\d+)\.	TABLE $1.	L...	L...	0	459	640	0	0	100		2	1	def...		87
0006	LOI Page References	(\d+)\s	PAGE $1	L...	L...	400	800	640	0	0	100		2	1	def...		88
0007	Paragraph Destinati...	(^\s)(\d(\.\d+)+) ([A-...	$2	B...	E...	0	0	0	0	0	100		1	1	def...		89
0008	Figure Destinations	(^\s)(FIGURE \d+\.)\...	$2	B...	E...	0	0	0	0	0	100		1	1	def...		90
0009	Table Destinations	(^\s)(TABLE \d+\.) (...	$2	B...	E...	0	0	0	0	0	100		1	1	def...		91
0010	PAGE Destinations	([\divx]+)$	PAGE $1	C...	E...	0	100	640	0	0	100		1	1	def...		92
0011	Body Figure Refere...	([Ff]igure)(s?)\s(\d+)	$1 $3.	B...	E...	0	0	0	0	0	100		2	1	def...		93
0012	Body Figure Refere...	and (\d+)\)	FIGURE $1.	B...	E...	0	0	0	0	0	100		2	1	def...		94
0013	Body Table Referen...	able (\d+)	TABLE $1.	B...	E...	0	0	0	0	0	100		2	1	def...		97
0014	Body Paragraph Ref...	([a-z]) (\d(\.\d+)+)	$2	B...	E...	0	0	0	0	0	100		2	1	def...		98

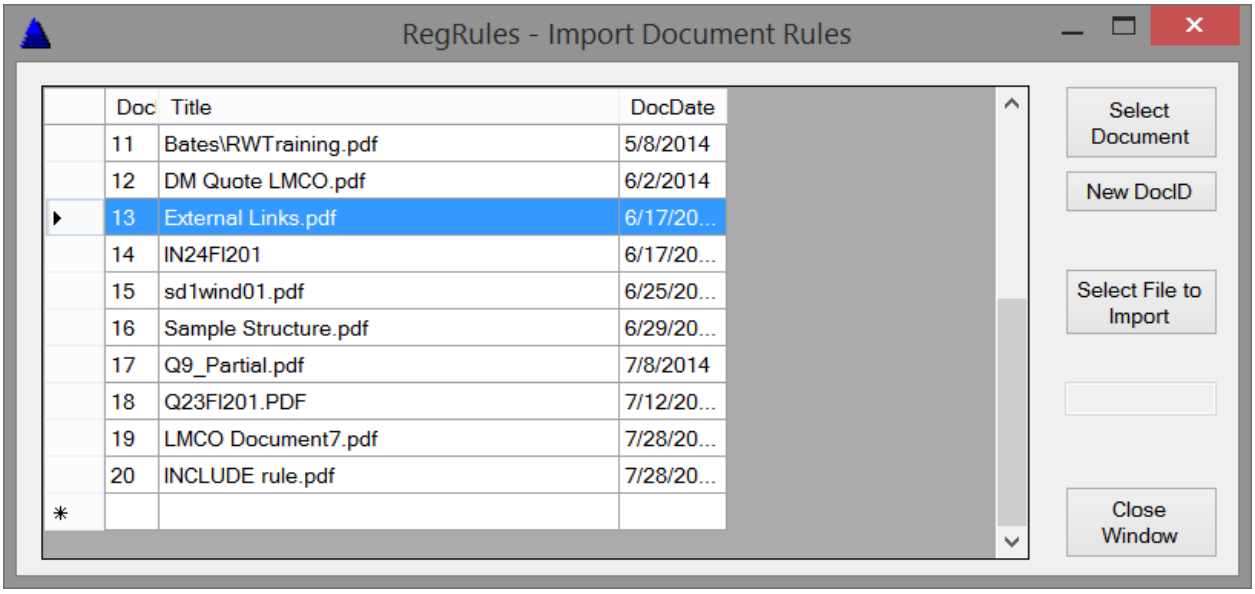
The user can select some of these rules using standard windows selection of clicks and control or alt key. The selected rules can be copied to an existing document using the "Copy to" button. The user selects the document of the destination for the rules and copies of the selected rules are appended to any existing rules for that document. The DocID for a document is the left field at the top right corner of the Rules Editor. In figure 13 the DocID is 20. In figure 1 the DocID is 1.

Figure 20. InfoLinker Plus Rules Copy Function.



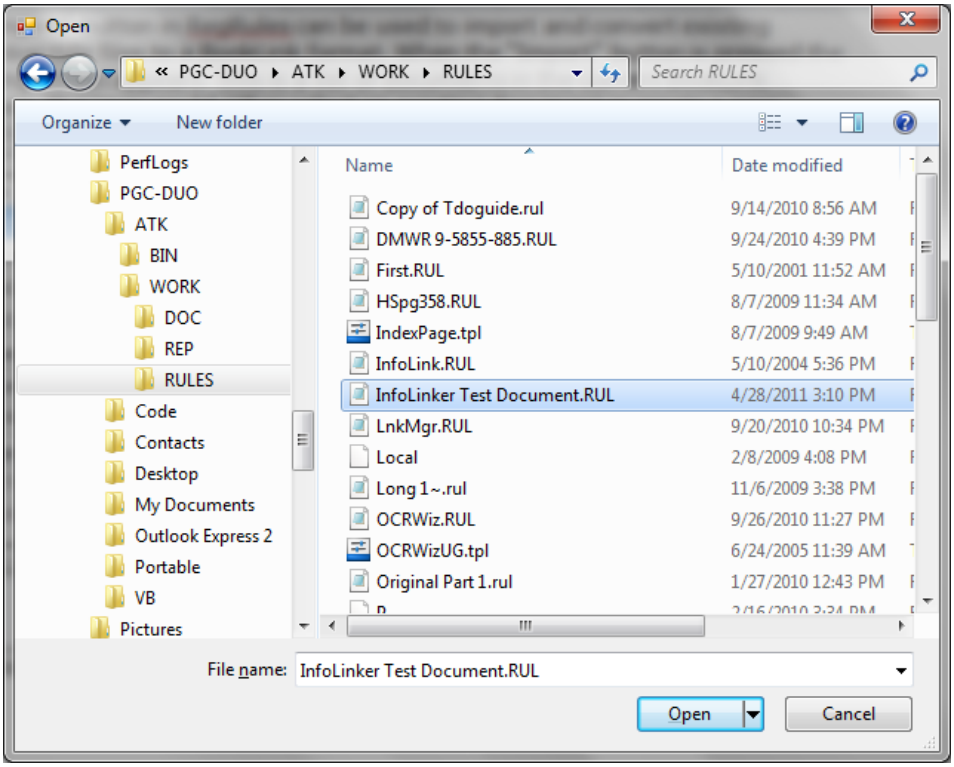
The "Importing" button in RegRules can be used to import and convert existing InfoLinker rule files to an ***InfoLinker Plus*** format. When the "Importing" button is pressed the following screen appears. This is a list of documents in the current ***InfoLinker Plus*** database. The user would either select an existing document or create a new document using the "New" button. After selecting the document to import InfoLinker rules press the "Select File to Import" button.

Figure 21. InfoLinker Plus Rules InfoLinker Import.



The "Select File to Import" button will display a file selection dialog to choose the rule file to import. Then select "Open". The rules will be imported as best they can and these rules will be available in the Rules Editor.

Figure 22. InfoLinker Rule File Selection.



Chapter 5: RegExp Interactive

InfoLinker Plus Compiler and Linker

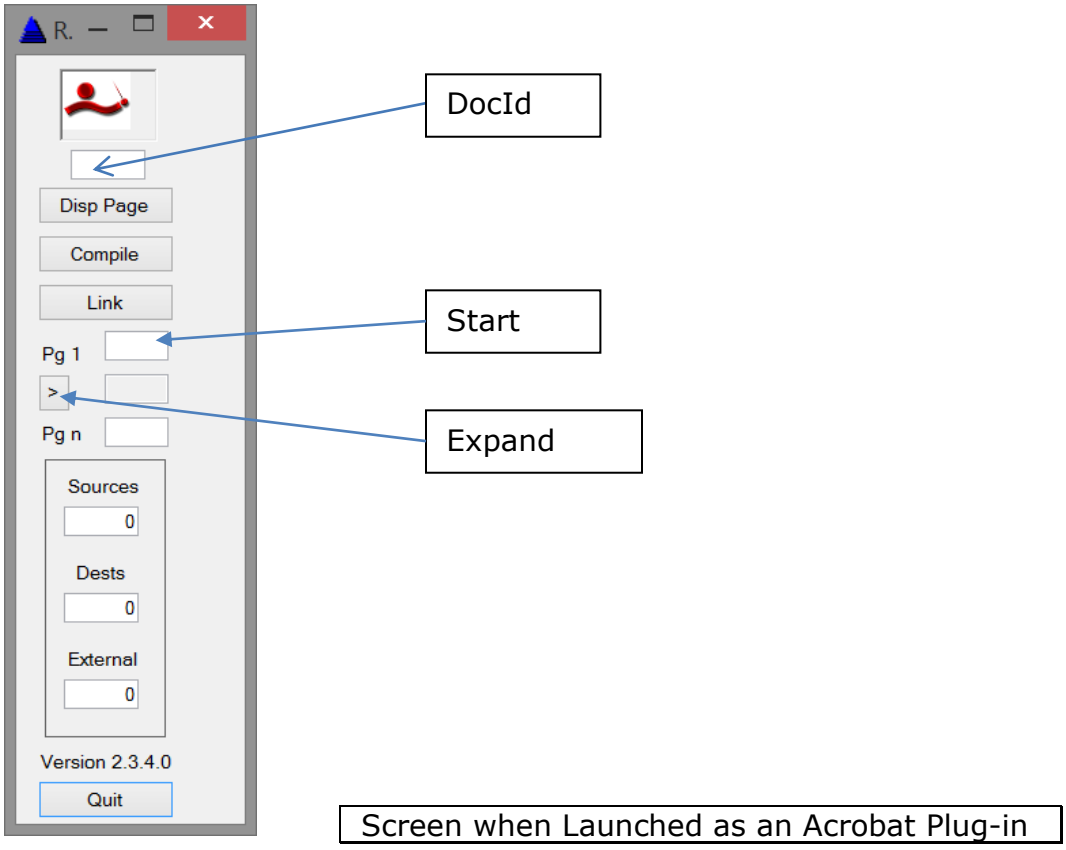
RegExp is a good program for learning how to write and test regular expressions and rename strings. RegExp is the program that *InfoLinker Plus* uses to process rules against the contents of PDF document. There are several steps that occur during the processing of a PDF document. The first step is to determine if there are any dynamic symbols to define. The dynamic symbol rule requires that the RegExp program reads the entire document's contents and looks for the page number of the symbols to be defined. As an example, if the user wants to use the page number of the "Table of Contents" in determining links, the program finds the string "Table of Contents" and sets the symbol to the first page number that contains the string. This symbol name can be used in other rules in the documents rule set.

The next step is to process each source and destination rule against each page in the PDF to determine any potential links in the document. This is called the compile. The next step is to determine from all of the potential links, which ones are valid and how are they named and organized. This is called linking. These steps of compiling and linking are the functions of the RegExp program.

RegExp can be launched from the Adobe Acrobat® *Plug-Ins* > *InfoLinker Plus* > *ILP Process* menu or as a stand-alone application (typically at C:\Program Files *(x86)*\Docmaestro\ILP\RegExp.exe or C:\Program Files\ISPAinc\ILP\RegExp.exe).

When you run the *InfoLinker Plus* > *ILP Process* menu, RegExp launches as a plug-in and runs your full rule set against the current document. When run as a plug-in, RegExp looks like this (figure 23).

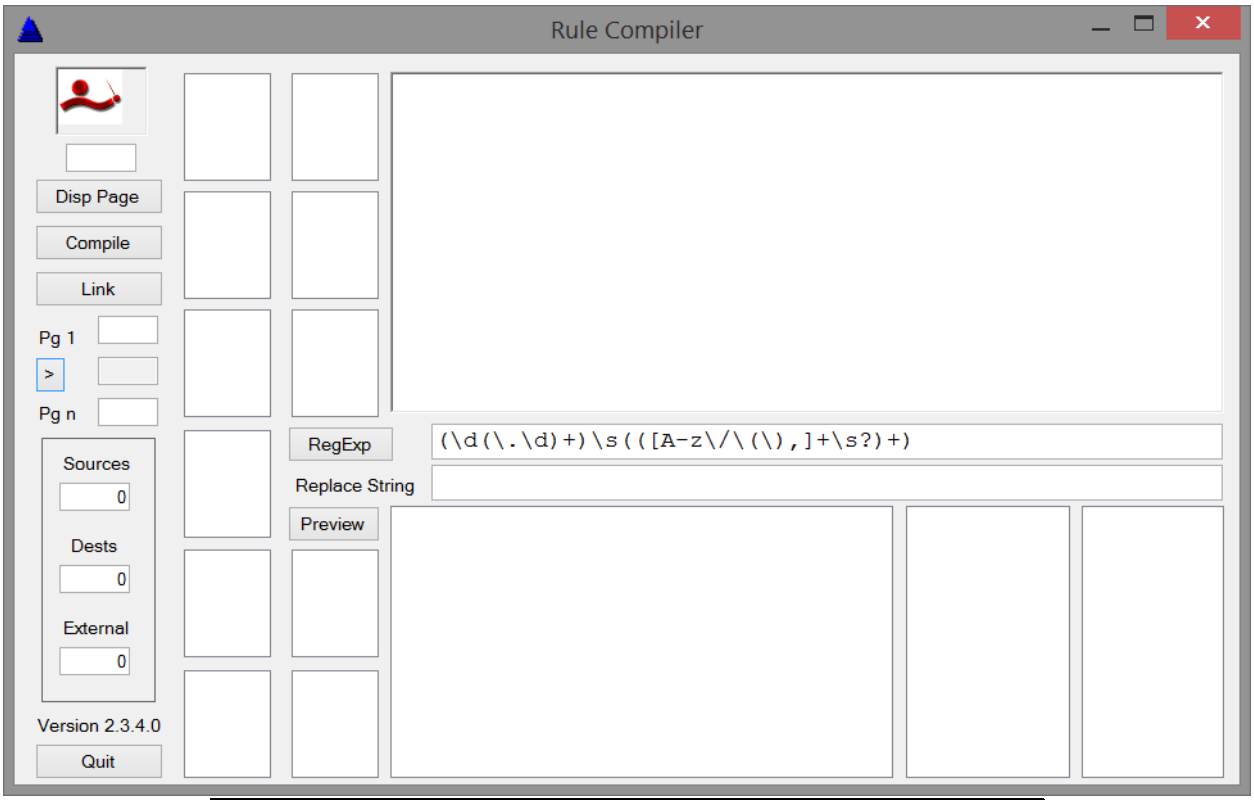
Figure 23. RegExp with InfoLinker Plus.



RegExp - A Regular Expression Tool

When you run RegExp as a stand-alone program by directly launching RegExp.exe from the application directory, RegExp has options that allow you to enter and test individual rule expressions. Pressing the Expand Screen "**>**" button displays additional information as shown in below (figure 24).

Figure 24. RegExp Interactive Mode.

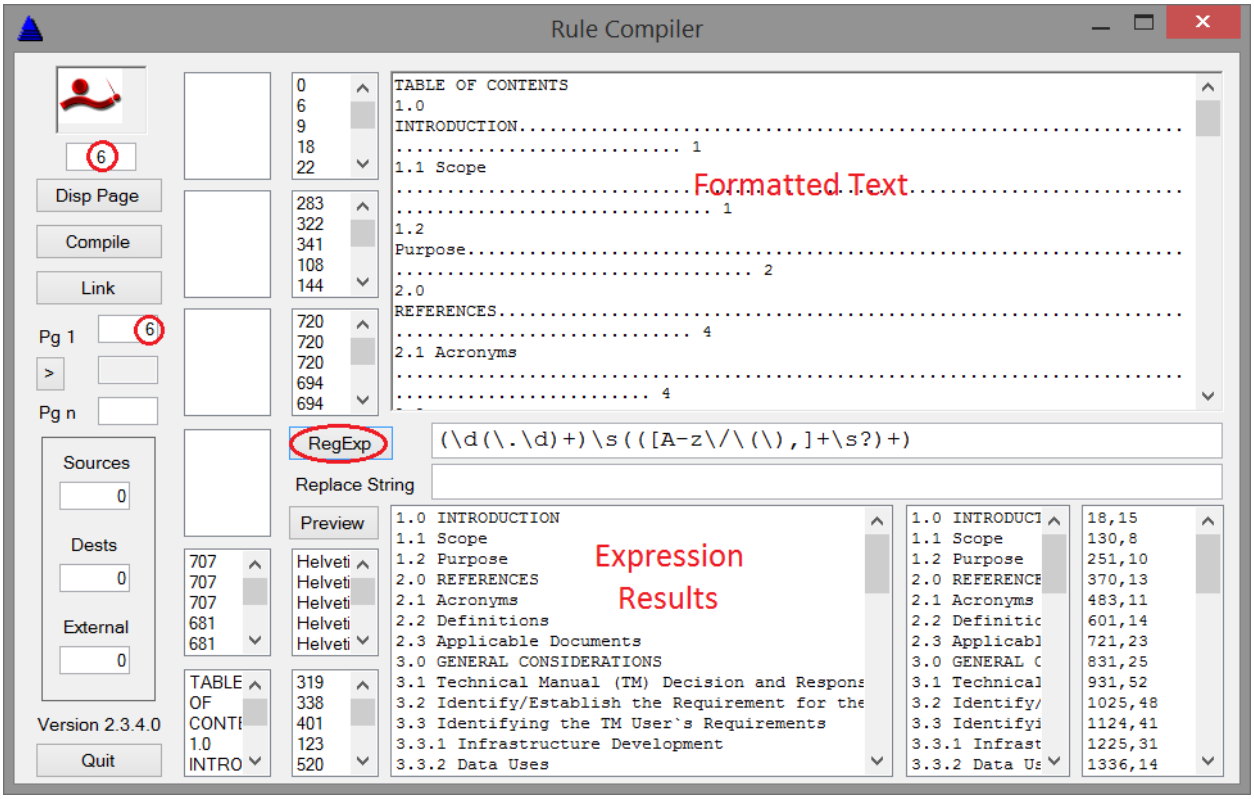


Screen when Launched as a stand-alone Application

The screen below (figure 25) shows PDF page 6 of DocID 6. This is the Sec7.pdf document. The upper right box displays the text as formatted from the PDF page. The content is created using the interpreted line feeds, carriage returns, tabs, spaces, and text from the PDF page. To the right of the "RegExp" button is the regular expression that is being tested against the current PDF page. You can modify the expression and test it by pressing the "RegExp" button. The test results are displayed in the lower right box.

The other boxes aren't too important for use in understanding how the compiler works. The boxes are useful in determining geometry and font information related to the PDF page text.

Figure 25. RegExp Display Screen.



If you want to try using a replacement string for the regular expression, you can place it in the text box to the right of "Replace String". Then press the "RegExp" button and the compiler will display the results in the box to the lower right.

Figure 26. RegExp Display Screen with Replacement.

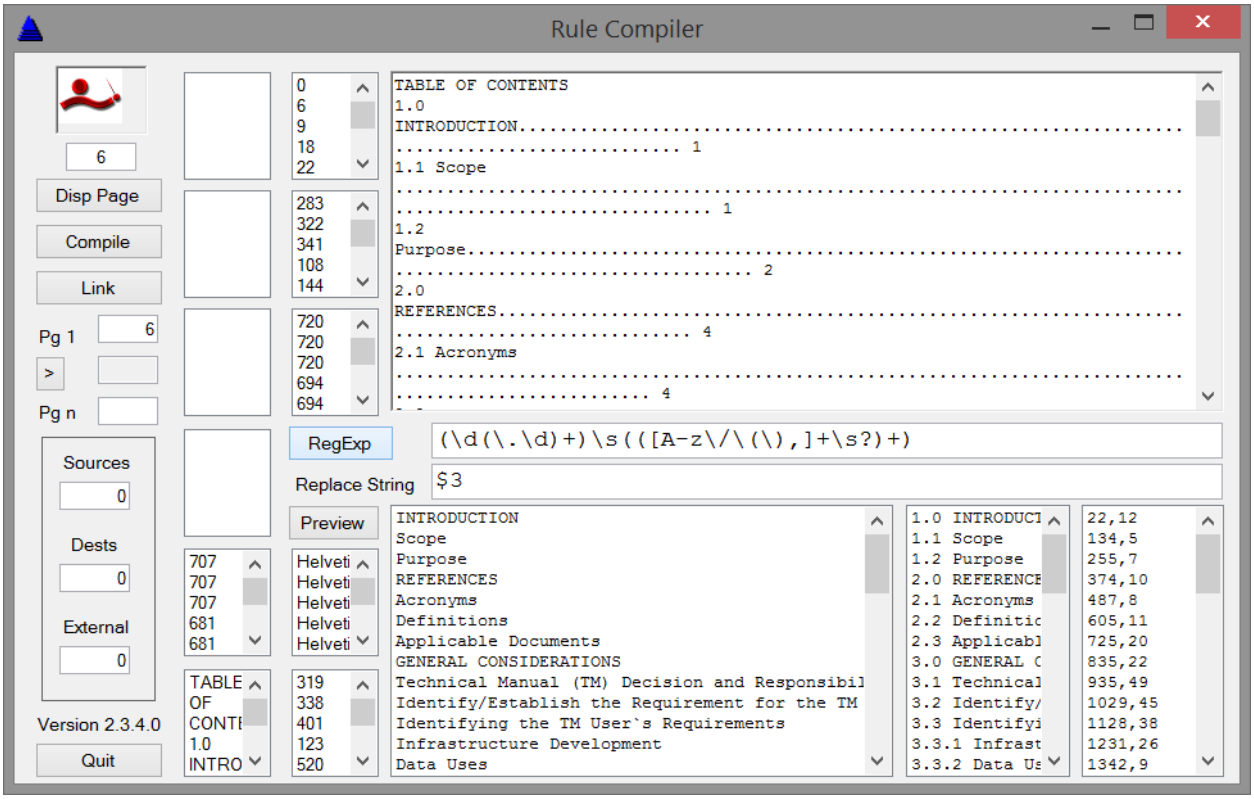
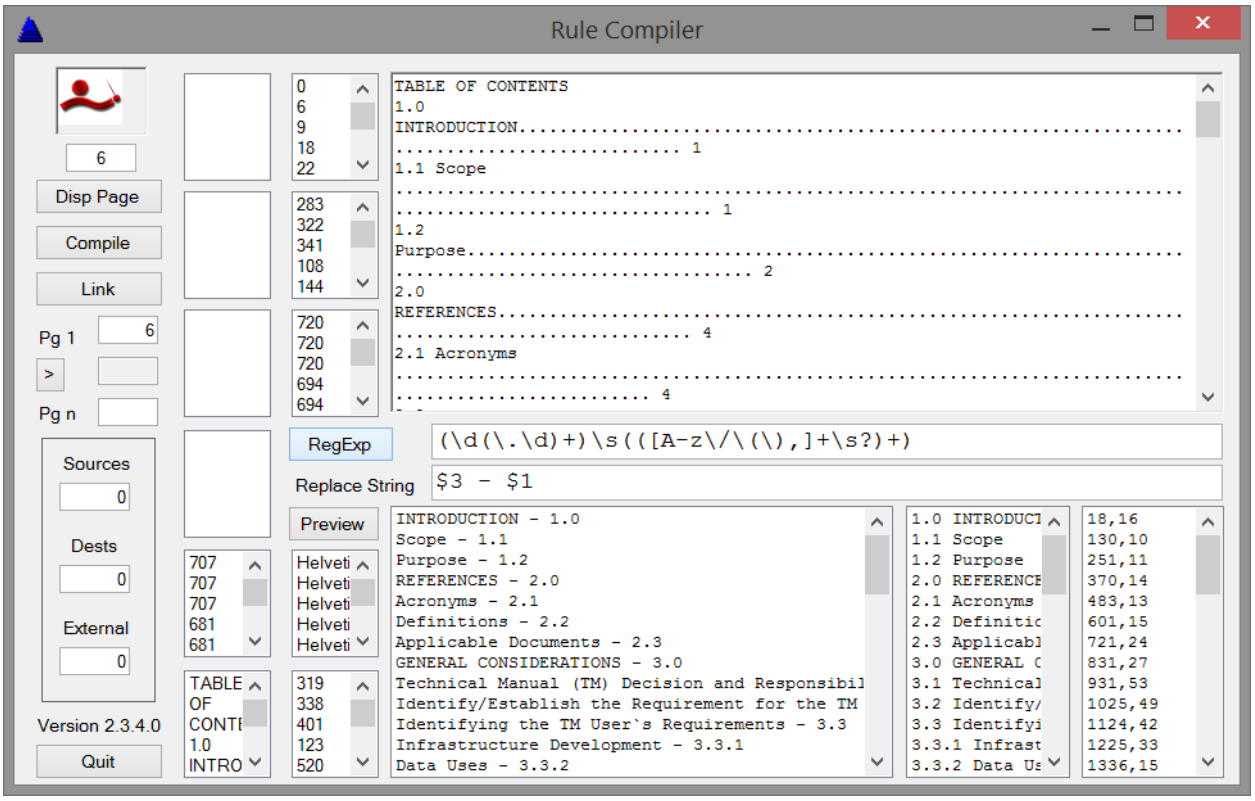


Figure 27 is an example using a different replacement string with the same regular expression against the same page of the PDF.

If you need to find the DocID of a document, it is displayed in the upper right of the Rules Editor screen. There are 2 unlabeled boxes in the upper right of that screen. The box on the right is the current RuleID and the one to the left is the DocID of the current Acrobat document. (See figure 13)

Figure 27. RegExp Interactive Example.

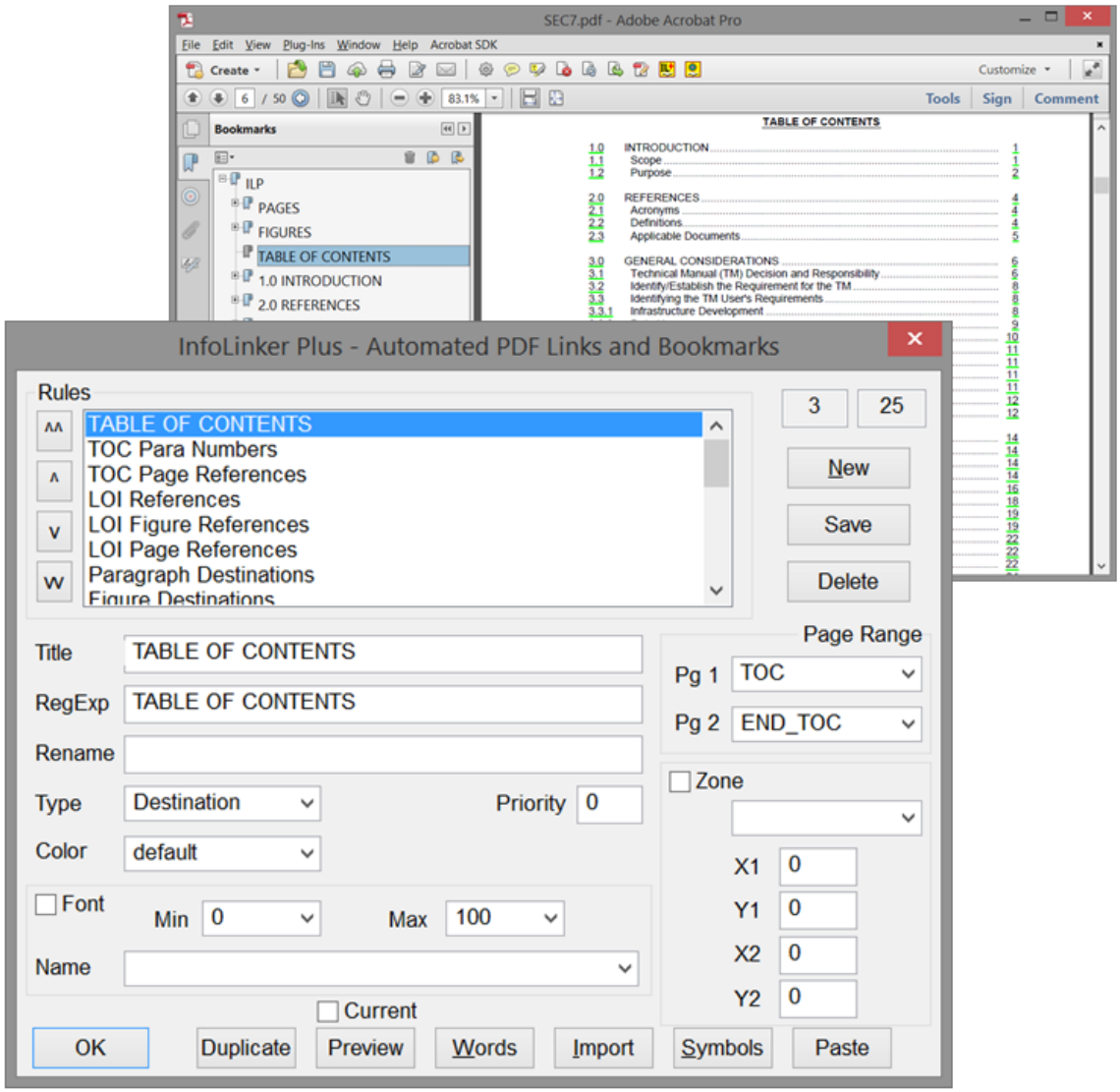


Chapter 6: InfoLinker Plus Sample Rules

Table of Contents Rule

The highlighted rule in the Rules Editor is a rule for building the "Table of Contents" bookmark in this sample document. This rule has a title "TABLE OF CONTENTS" and the search expression (RegExp) is "TABLE OF CONTENTS". The page range that the rule searches is TOC to END_TOC. The rule type is "Destination". When the expression is found, ***InfoLinker Plus*** creates a bookmark called "TABLE OF CONTENTS" and places it in the "Bookmarks" tree of the PDF. There is no zone nor font to better qualify the search expression. The "TABLE OF CONTENTS" string is underlined in the default link color of gray.

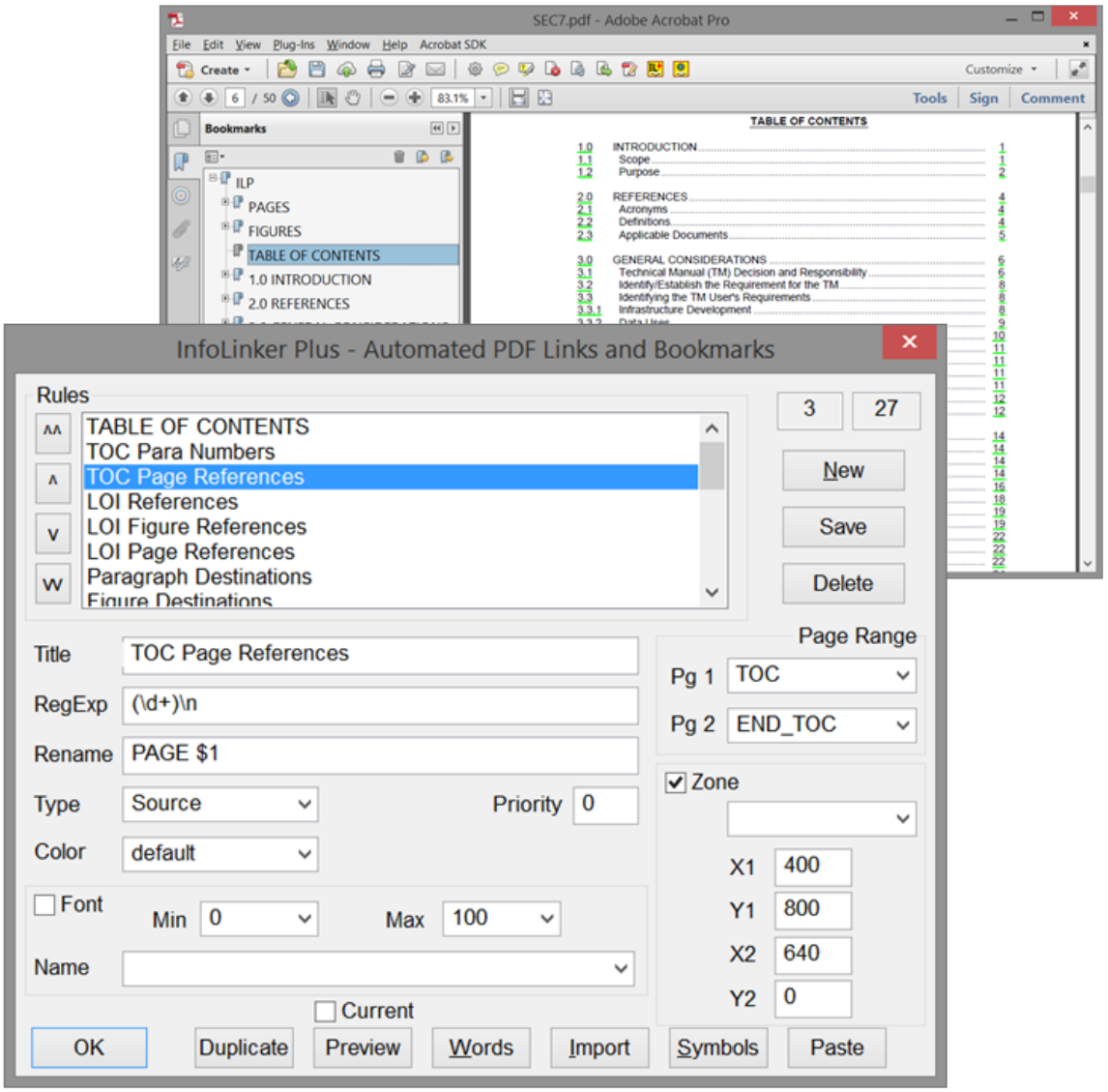
Figure 28. Table of Contents Rule.



Page Reference from the Table of Contents Rule

This rule is titled "TOC Page References". The page range that the rule is applied is from the TOC page to the END_TOC page. The search pattern (RegExp) that the rule looks for is "**(\d+)\n**". This is described as any sequence of numeric characters "**(\d)+**" that end with a carriage return "**\n**". All the numbers at the end of a line in the table of contents are references to page number in the body of the document. These are Source rules that create links to a view of the page name that is referenced. This rule uses a zone qualifier to look only at the right hand side of the page. The zone, in this case, will not create a link from the page number at the bottom middle of the page.

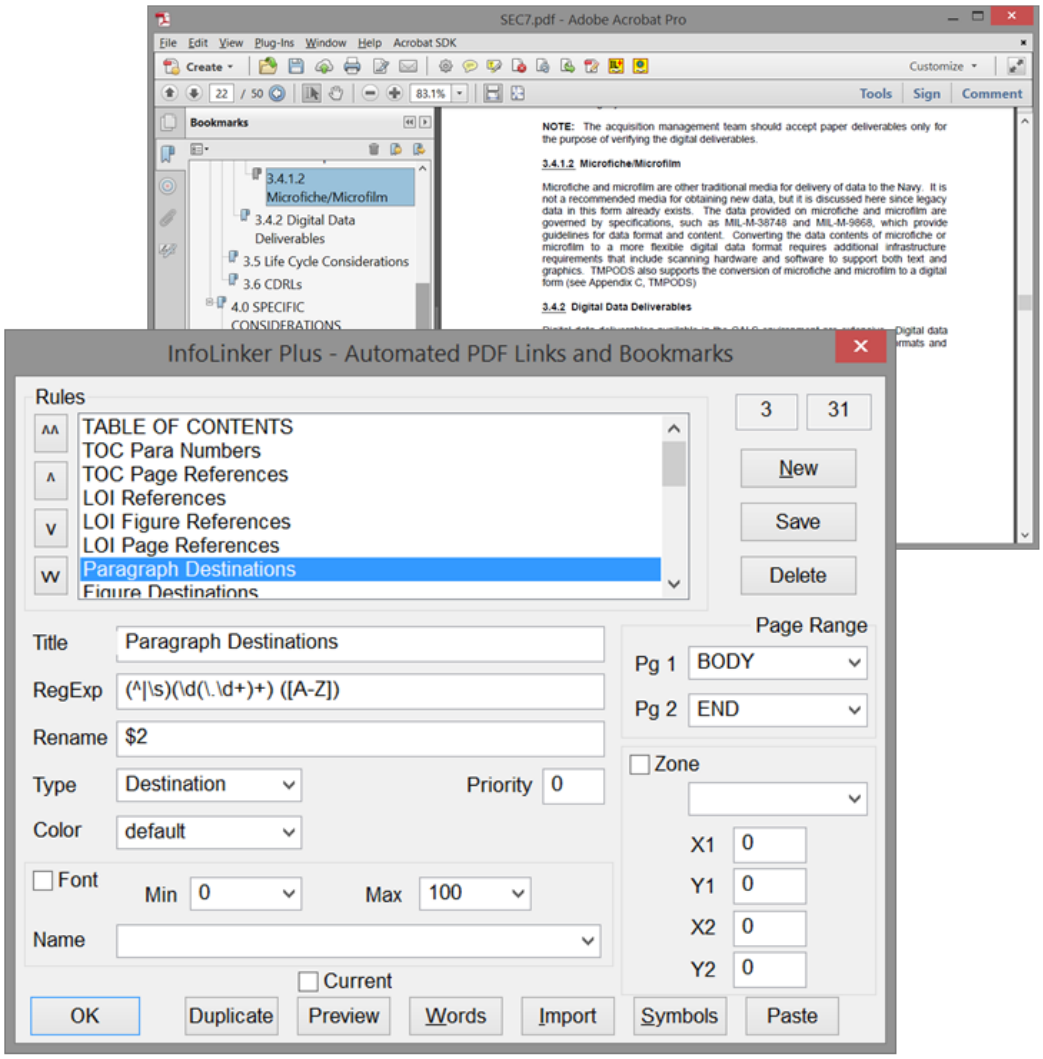
Figure 29. Page Reference Rule.



Paragraph References in the Table of Contents Rule

This document has paragraph naming using a number-dot-number pattern (see figure 30). The rule applies to the body of the document, BODY to END. The search pattern (RegExp) is looking for the beginning of page or beginning of a line "**(^|\s)**" to qualify the search pattern. The pattern continues with a number "**\d**" followed by a repeatable pattern of a period followed by one or more numbers "**(\.\d+)+**". This must be followed by a space and an uppercase letter "**([A-Z])**". The second part of the search pattern (based on the sets of parenthesis is used as the name of the bookmark and destination name. The result string of "**(\d(\.\d+)+**" (assuming all other conditions are met) is used as the name. In the figure below the bookmark **3.4.1.2** is the bookmark that meets the requirement of this rule. The string "**Microfiche/Microfilm**" is added to the bookmark display name because ***InfoLinker Plus*** can optionally add any characters following a bookmark name that are on the same line and have the same font all the way to the end of the line. This is a Destination rule.

Figure 30. Paragraph Reference Rule.



Page Destination Rule

The rule that finds the page names of each page in the document is highlighted in the figure below. This rule is titled "PAGE Destinations". The rule is applied to every page in the document, COVER to END. If the document only had page "numbers" then the search pattern would be "**\d+**", but this document has page names that use roman numerals, so there are a set of characters that could be part of the page name. This set includes numbers and the "I", "v", and "x" characters. The set is "**[\divx]**" and it is repeatable, defined by "**([\divx]+)**". The "**$**" symbol is used to represent the end of the page. So, any number of roman numeral sequence at the end of the page defines the page name in this document. We rename the page name with the string "PAGE" followed by the pattern found at the end of the page. Sample bookmark names could be "PAGE 1", "PAGE 12", "PAGE 225", and "PAGE xvi". There is also a zone of the page that the rule is restricted to find patterns. The zone (X1=0, Y1=100, X2=640, Y2=0) is the full width of the bottom of a page that is 8 ½ by 11 inches.

Figure 31. Page Destination Rule.

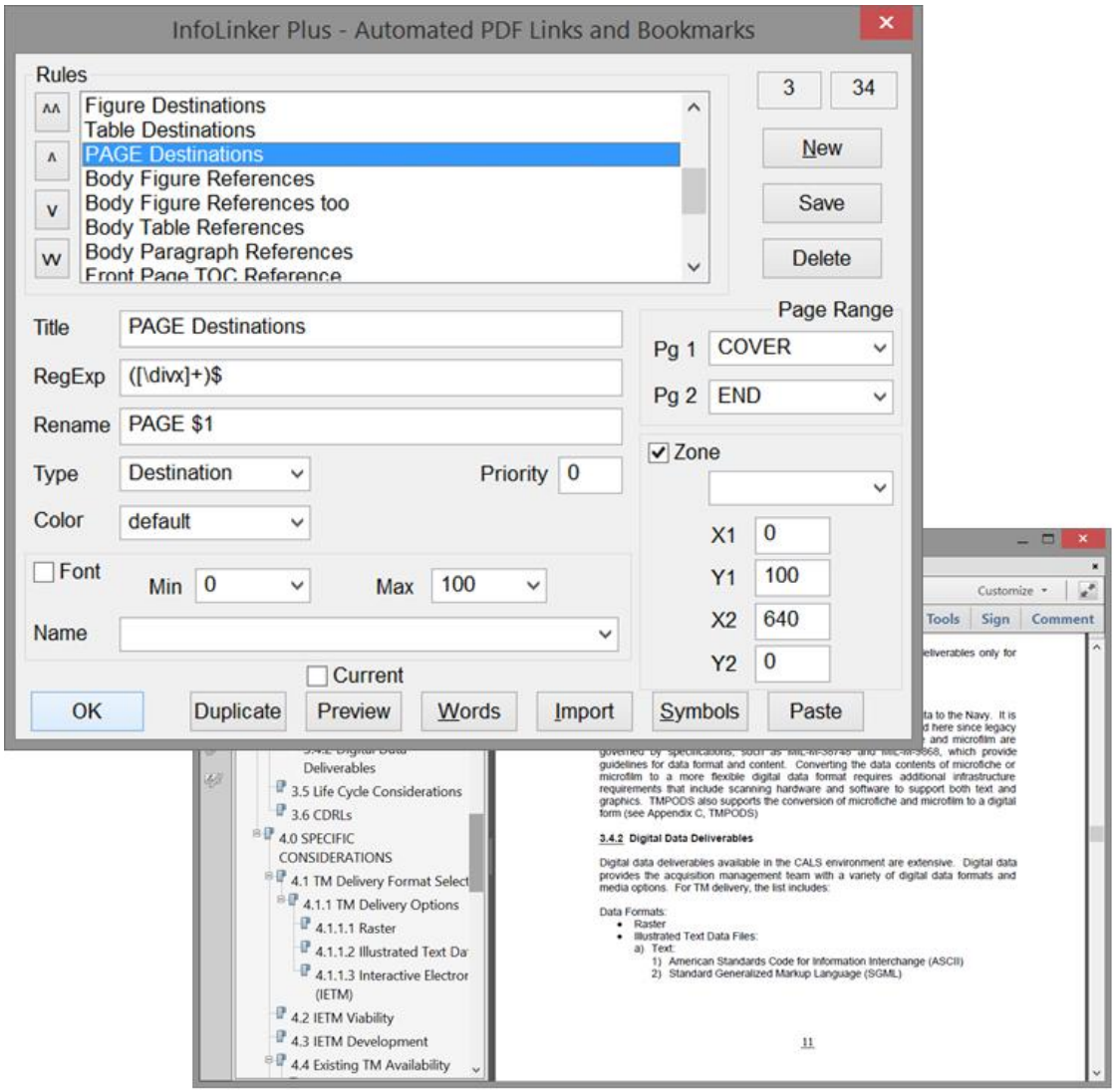
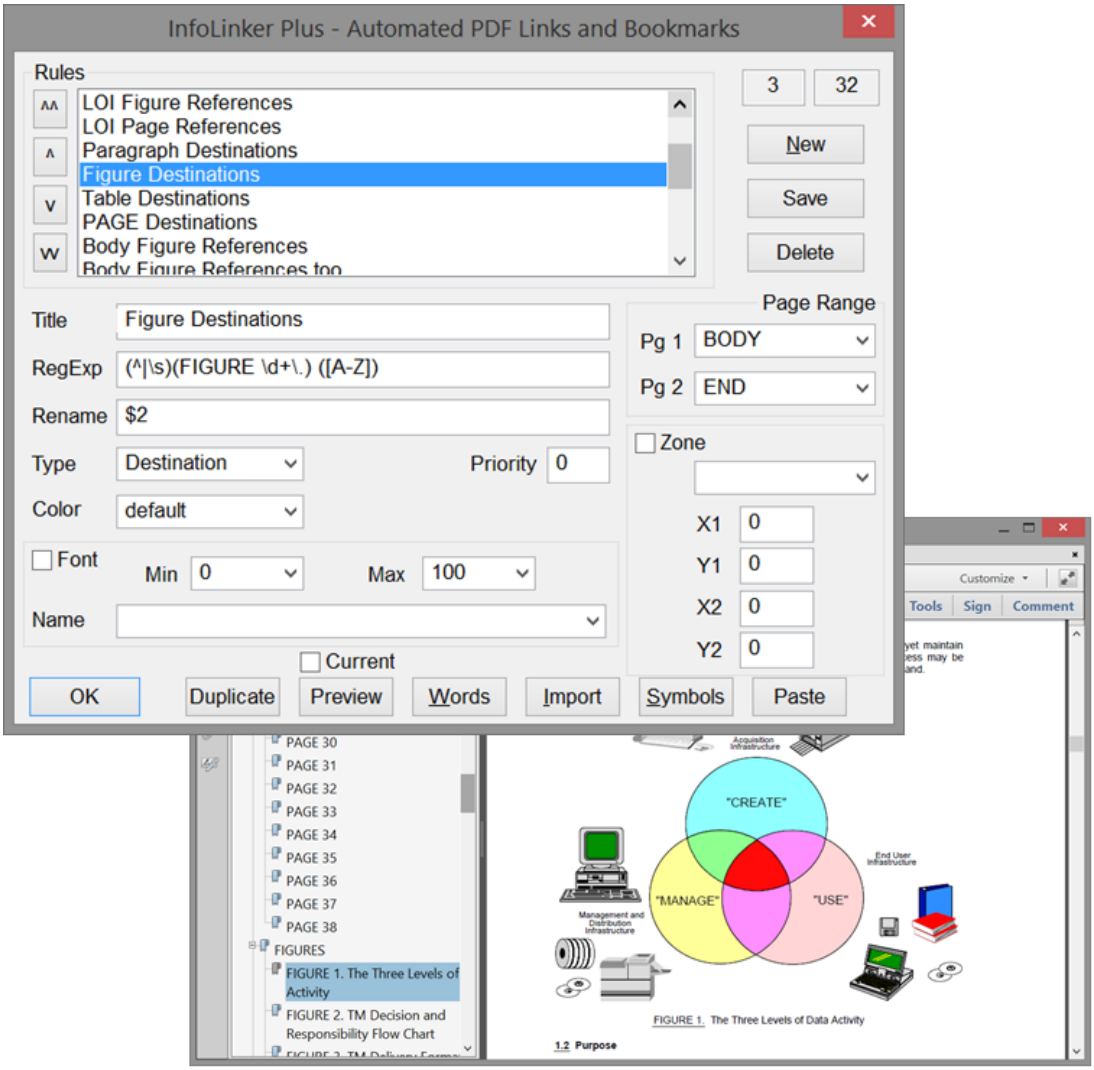


Figure Destinations Rule

The next rule looks for all of the figures in the body of the document. The title of the rule in the figure is "Figure Destinations". It is a Destination type of rule. It applies to the body of the document. And the pattern that is searched (RegExp) is the start of a string "**(^|\s)**", then the word "**FIGURE**" followed by a space a number and a period. This pattern is then followed by another space and an uppercase letter "**[A-Z]**". The second search pattern in parenthesis defines the name of the destination. That is "**(FIGURE \d+\.)**" defines the destination name. Examples are "**FIGURE 1.**", "**FIGURE 4.**", and "**FIGURE 12.**". The bookmarks that are created also added the text string that follows the destination name that is on the same line of text. Example: "**FIGURE 1. The Three Levels of Activity**". Any source links to this figure only need to have the pattern of "Figure 1." Or "FIGURE 1." to link to this object.

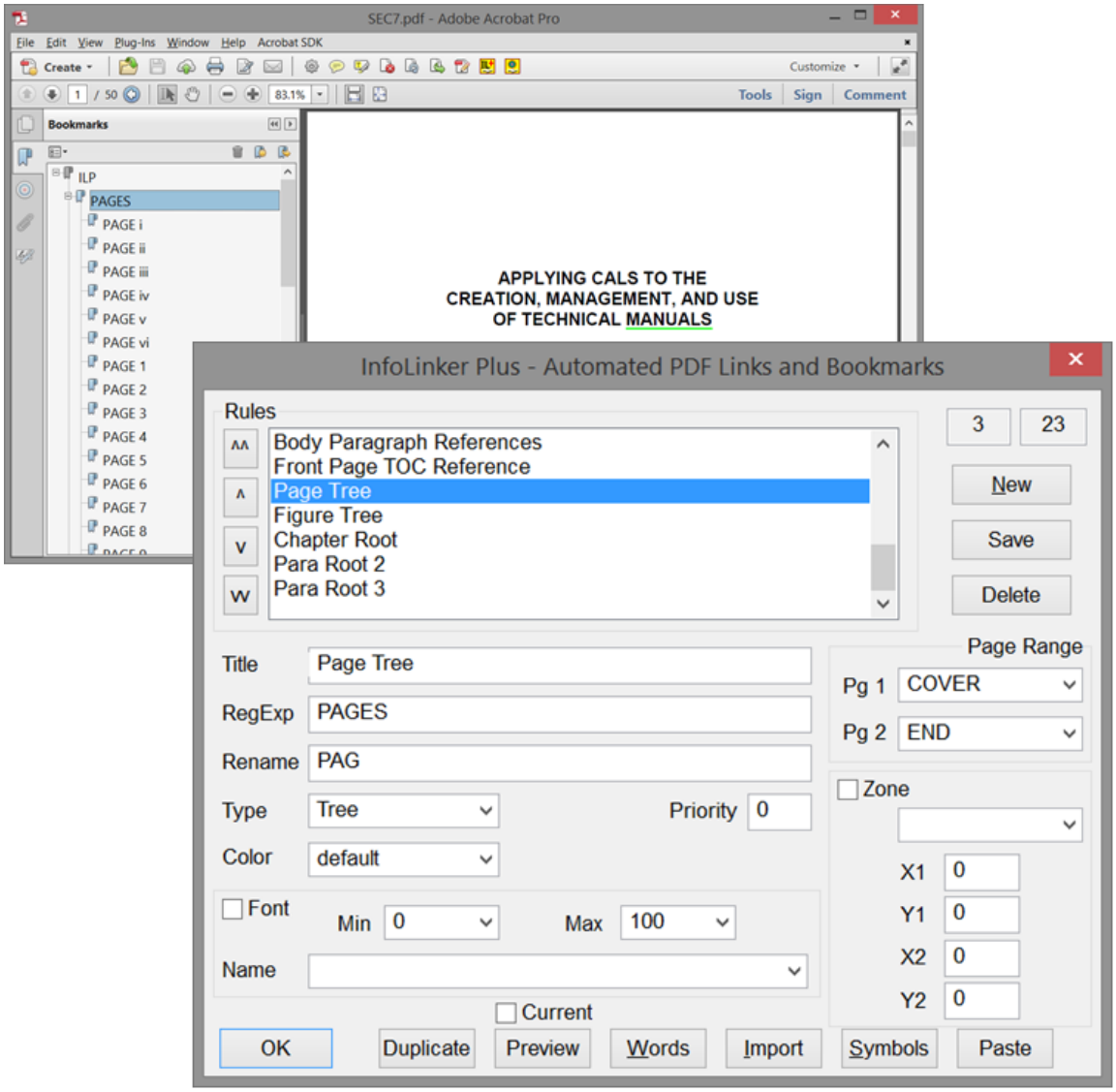
Figure 32. Figure Destination Rule.



Tree Rule

Some of the rule types are used to manipulate the bookmarks created by ***InfoLinker Plus***. The Tree Rule Type is used to collect similar bookmarks and nest them under a new or existing bookmark. The following rule is titled "Page Tree". This rule is compared to all bookmarks created for this sample document. If a bookmark matches the search pattern (RegExp) then the bookmark is nested under a bookmark named "**PAGES**". If the bookmark "**PAGES**" does not exist then it is created. The search pattern used by this rule is "**PAG**". If any bookmark contains "**PAG**" then it is nested under "**PAGES**". All of the page name bookmarks are nested under the root bookmark "**PAGES**".

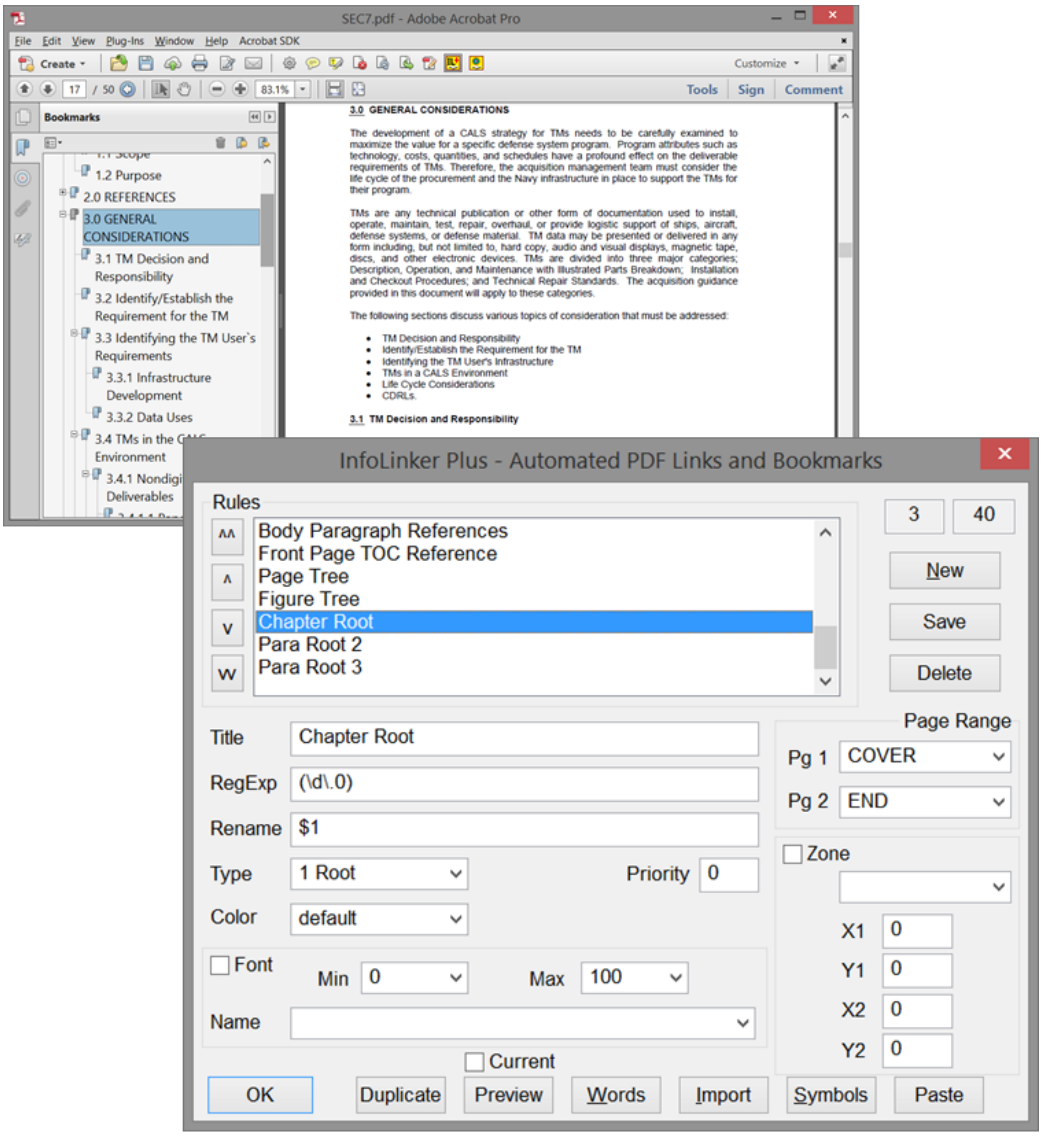
Figure 33. Tree Rule.



Chapter Root Rule

The root rule type is a bit complicated but very powerful. Figure 34 is an example of a rule that will nest existing bookmarks based on the search pattern in the rule. Every bookmark that follows the root bookmark that matches the root rule will get nested under that bookmark. This rule applies to all of the bookmarks that ***InfoLinker Plus*** created in the document. In this example if a bookmark is a number followed by a period and a zero "**\d\.0**" then any bookmark following that matching bookmark will get nested under the match.

Figure 34. Chapter Root Rule.



The bookmarks will continue to get nested until another bookmark matches the root pattern and then it becomes the new root that other bookmarks will get nested under. The bookmarks are compared in page order. In this sample PDF, the first bookmark for a paragraph in a chapter is a number followed by a period and a zero.

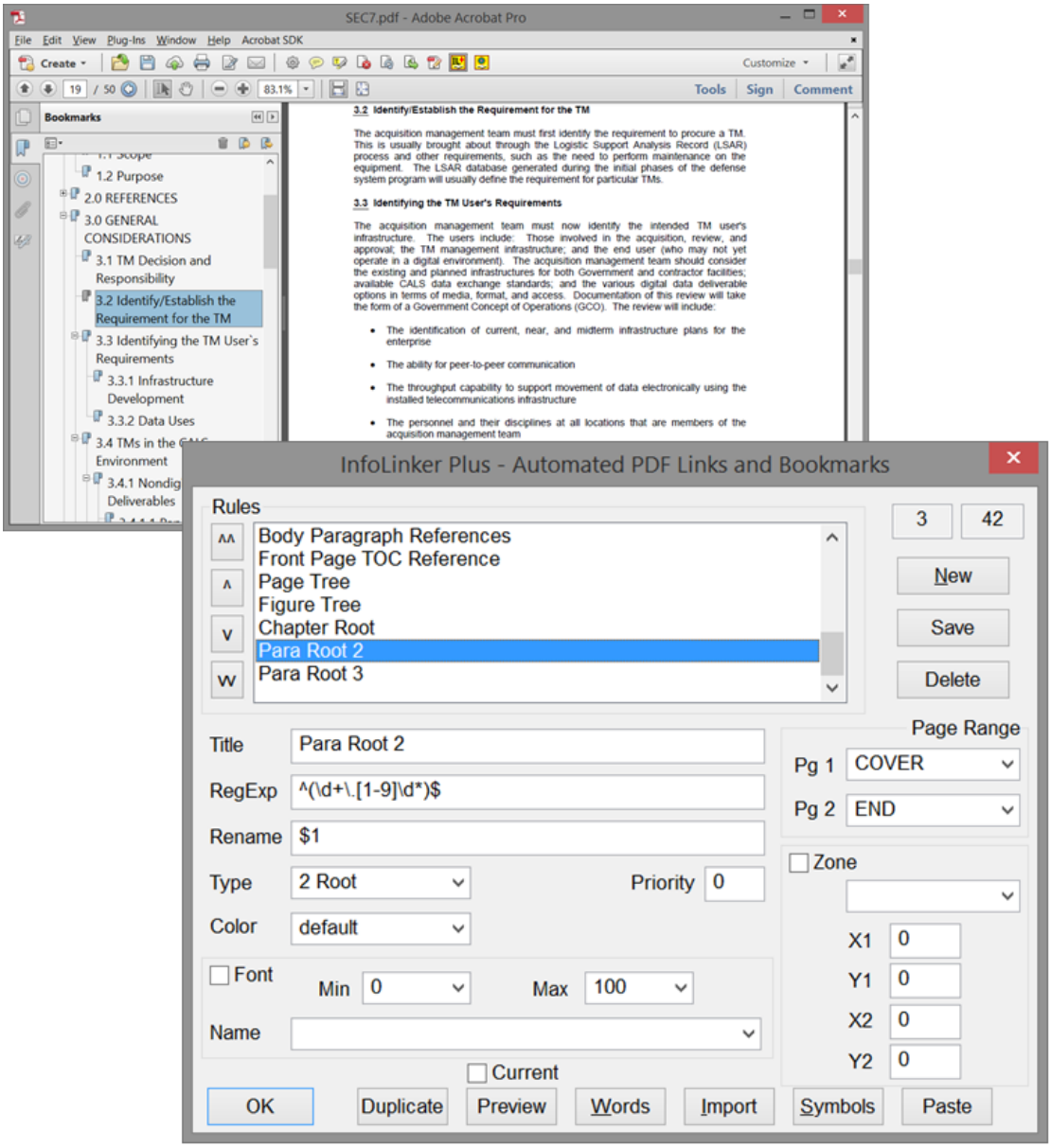
All of the following paragraphs in that chapter start with the chapter number and then sequentially have period and number sequences in their name. The Root rules are applied after all of the Tree rules are applied to the set of bookmarks so all of the tree bookmarks are not nested under the root bookmarks. If there are no root bookmarks then all bookmarks could get nested under a root bookmark.

Paragraph Root 2 Rule

The Root Type Rule 2 (figure 35) can be used to nest other sets and sequences. This sample document has paragraph naming sub sections inside paragraphs. The first paragraph in a chapter is, for example, "**1.0**" and the next paragraph is the beginning of the next section titles "**1.1**". There can be subsections that get named using another appended pattern like "**1.1.1**". Since the document is consistent we can use Root 2 Rule Type to nest the chapters as well as subsections.

This rule looks at the destination names and if a pattern is found then the following bookmarks will get nested under the found root. The search pattern is beginning of name "**^**" followed by a number sequence "**\d+**", a period "**\.**", a digit (not zero) "**[1-9]**", an optional additional digit "**\d***" and then the end of the name "**$**". The name (not the beginning of the string nor the end of the string) becomes the new root that other bookmarks are nested under until another root is found.

Figure 35. Paragraph Root 2 Rule.



Appendix A: License Agreements

This section provides DocMaestro, LLC's Software License Agreement, Total Support Agreement, and Software Maintenance Agreement.

DocMaestro Software License Agreement

Please read this software license agreement ("License") carefully before installing the software. By installing the software, you ("User") are agreeing to be bound by the terms of this license. If you do not agree to the terms of this license, do not install the software. The software product and version to which this License applies is annotated at the bottom of this License. The User should print this License, and the License number contained on the product packaging should also be annotated, for User's records.

License. This is an agreement between Licensor (DocMaestro, LLC) and Licensee, who is being licensed to use the named Software Product. Licensee acknowledges that this is only a limited nonexclusive license. Licensor is and remains the owner of all titles, rights, and interests in the Software. The software accompanying this Agreement - whether on disk, in read only memory, on any other media or in any other form ("DocMaestro Software") – are Licensed, not sold, to User by DocMaestro, LLC. User owns the media on which the DocMaestro Software is recorded but DocMaestro retains title to the DocMaestro Software. The DocMaestro Software and any copies made and/or distributed under this License are subject to the terms and conditions of this License Agreement.

Permitted Uses and Restrictions. This software is distributed via download and/or on a CD-ROM under a License Agreement with DocMaestro. Use of this software is restricted to accessing the documents distributed via download and/or on the CD-ROM set delivered with this software, and supplemental documents provided from the subscription provider. This License allows User to use the DocMaestro Software on one PC. User may make one copy of the DocMaestro Software in machine-readable form for backup purposes only. The backup copy must include all copyright information contained on the original. Except as permitted by applicable law and this License, User may not de-compile, reverse engineer, disassemble, modify, rent, lease, loan, distribute, create derivative works from the DocMaestro Software or transmit the DocMaestro Software over a network. User may, however, transfer your rights under this License provided User transfers the related documentation, this License and a copy of the DocMaestro Software to a party who agrees to accept the terms of this License and destroy any other copies of the DocMaestro Software in your possession. User rights under this License will terminate automatically without notice from DocMaestro if User fails to comply with any term(s) of this License.

This Software is subject to a limited warranty. Licensor warrants to Licensee that the physical medium on which this Software is distributed is free from defects in materials and workmanship under normal use, the Software will perform according to its printed documentation, and to the best of Licensor's knowledge. Licensee's use of this Software according to the printed documentation is not an infringement of any third party's intellectual property rights. This limited warranty lasts for a period of 90 days after delivery. To the extent permitted by law, THE ABOVE-STATED LIMITED WARRANTY REPLACES ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, AND LICENSOR DISCLAIMS ALL IMPLIED WARRANTIES INCLUDING ANY IMPLIED WARRANTY OF TITLE, MERCHANTABILITY, NONINFRINGEMENT, OR OF FITNESS FOR A PARTICULAR PURPOSE. No agent of Licensor – current or past –, contractor, consultant, etc. is authorized to make any other warranties or to modify this limited warranty. Any action for breach of this limited warranty must be commenced within one year of the expiration of the warranty. Since some jurisdictions do not allow any limit on the length of an implied warranty, the above limitation may not apply to this Licensee in those jurisdictions. If the law does not allow disclaimer of implied warranties, then any implied warranty is limited to 90 days after delivery of the Software to Licensee. Licensee has specific legal rights pursuant to this warranty and, depending on Licensee's jurisdiction, may have additional rights.

Limitation of Liability. UNDER NO CIRCUMSTANCES, INCLUDING NEGLIGENCE, SHALL DocMaestro BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO THIS LICENSE. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THIS LIMITATION MAY NOT APPLY TO USER. In no event shall DocMaestro total liability to User for all damages exceed the amount of fifty dollars ($50.00).

Export Law Assurances. Users may not use or otherwise export or re-export the DocMaestro Software except as authorized by United States law and the laws of the jurisdiction in which the DocMaestro Software was obtained. In particular, but without limitation, the DocMaestro Software may not be used or otherwise exported or re-exported (i) into (or to a national or resident of) any United States embargoed country or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals or the U.S. Department of Commerce's Table of Denial Orders. By using the DocMaestro Software, User represents and warrants that User is not located in, under control of, or a national or resident of any such country or on any such list.

Government End Users. If the DocMaestro Software is supplied to the United States Government, the DocMaestro Software is classified as "restricted computer software" as defined in clause 52.227-19 of the FAR. The United States Government's rights to the DocMaestro Software are as provided in clause 52.227-19 of the FAR.

Controlling Law and Severability. This License shall be governed by the laws of the United States and the State of Georgia. If for any reason a court of competent jurisdiction finds any provision, or portion thereof, to be unenforceable, the remainder of this License shall continue in full force and effect.

Licensee agrees to defend and indemnify Licensor and hold Licensor harmless from all claims, losses, damages, complaints, or expenses connected with or resulting from Licensee's business operations.

Licensor has the right to terminate this License Agreement and Licensee's right to use this Software upon any material breach by Licensee.

Licensee agrees to return to Licensor or to destroy all copies of the Software upon termination of the License.

Complete Agreement. This License constitutes the entire agreement between the parties with respect to the use of the DocMaestro Software and supersedes all prior or contemporaneous understandings regarding such subject matter. No amendment to, or modification of this License will be binding unless in writing and signed by DocMaestro.

Product: _____ Date Purchased: _____

Serial Number: _____

_____

Authorized User *(signature indicates acceptance of above terms and conditions)*

(Please sign and email back to DocMaestro. sales@docmaestro.com)

Signed for and on behalf of DocMaestro, LLC:

_____

Robert Radtke, Managing Partner **Date**

DocMaestro Total Support Agreement

This Total Support Agreement (Hereinafter referred to as TSA) between DocMaestro, and _____, ("User") shall be effective for a period of one calendar year. This Agreement entitles the "User" to services covering software and hardware maintenance, product upgrades, and technical support services as defined by the terms of this Agreement. Technical support services extend to the version, revision level, and maintenance releases for the solution(s) listed below, purchased by the "User" and licensed from DocMaestro, hereinafter referred to as "solution".

Total Support Agreement (TSA) will cover the following software and hardware**:**

_____

_____

_____

Terms of the Total Support Agreement (TSA)

Purchase of a TSA entitles the "user" to the following: free product upgrades of purchased solution(s), and a four (4) business hour* response time during normal business hours.

This agreement extends only to platforms and operating environments certified for use with the licensed product. DocMaestro is not responsible for integration or configuration with third-party software, hardware, or operating environments, unless otherwise defined within the Statement of Work. The agreement provides assurance to the "user" that the licensed software operates in conformity with terms set forth in the Software License Agreement that accompanied the licensed product at the time of purchase. Violation of your license will nullify this support agreement.

* Normal business hours of 8:00 AM to 6:00 PM EST, Monday through Friday excluding national and locally observed holidays.

Scope of Service

Upon joint agreement and approval of the solution configuration, including the design of Rule Templates, Forms for item routing and the Workflow process flowchart, "user" rights, system screens and any data transfer method to and from third party objects, if defined in the statement of work, the solution shall be considered as accepted.

Upon acceptance, DocMaestro will provide all necessary software upgrades, enhancement patches and will participate in any meetings or discussions with regard to modifications to the accepted solution. These modifications may include, for example, minor changes to the accepted workflow, forms, and/or templates. A system re-configuration, e.g. modifications made to operating environment, is not covered under the TSA.

The TSA covers one (1) coordinator level and one (1) requestor level refresher training class.

"User" Responsibilities

This support agreement extends only to the DocMaestro provided solution(s), free of additions or changes that have not been made or approved by DocMaestro Support requests are made to DocMaestro, LLC by submitting a Technical Support Request Form via www.pgcsolutions.com. A link to the form can be found in the "Products and Support" section of DocMaestro website.

The "user" should designate a Solution Administrator and provide contact details through which DocMaestro support staff can contact the appropriate customer staff or representative. This will normally be a telephone and fax number and/or an email address. Ensure that any fixes that are supplied as part of this service are not made available to any third party.

The "user" should provide adequate information to assist in verification and resolution of the support request. Failure to provide adequate information may result in billable charges for time spent investigating and duplicating the problem. Alterations to the solution database not directed under the guidance of DocMaestro that result in a database corruption will require an additional fee per support request. "User" or third party customizations to the solution that result in corruption of the product or data will require an additional fee per support request and may void the TSA. If during the resolution of any incident, it is determined that on-site assistance is required, the owner agrees to provide access to the system and authorized

technical personnel to assist DocMaestro support staff. Failure to provide the above mentioned access and/or staff may require an additional fee per support request.

DocMaestro Responsibilities

DocMaestro support technicians will aid in the resolution of support requests by contacting the "user" within 4 business hours of receipt of the Technical Support Request Form. All maintenance releases or revisions, including program fixes and related documentation will be made available to the "user" via the DocMaestro website. DocMaestro is not responsible for integration or configuration of third-party software, hardware, or operating environments.

Problem Resolution

All problems communicated via customer calls, will be investigated during normal working hours. In order to carry out an investigation of the problem, the support specialist will advise the customer of what supporting evidence, if any is required.

The anticipated solution for the problem (based on the results of the investigation), may be the provision of a known repair, or the generation of a repair to solve a new problem, or instruction on how to circumvent the problem. The resolution of the problem may be via remote connection to the customer's system using VPN or other Internet related access, email or direct dial depending on system availability.

Problem Escalation Management

In instances where a known repair is not available and a software repair/modification is required to resolve the problem, DocMaestro will discuss with the customer the severity of the impact of the solution error on their business, agree on a plan of resolution and record action plan.

On-Site Assistance

"User" will be permitted to invoke a site visit by a solution specialist where an incident cannot be completed within the operation of a telephone call or via remote connectivity onto the "user" system.

Service Limitations

DocMaestro cannot guarantee that all errors will be corrected, that fixes will be available, or that circumvention advice will be provided in all circumstances.

The service provided does not include the restoration of any lost data, resulting from "user" failure to secure the data in the system prior to making a call.

Exclusions to contract:

Onsite software support does not cover configuration issues that arise from the introduction of new hardware or software to the system. Steps required to remedy support issues raised by this action will be charged at our standard rate*

* Standard Rate for solution modifications, "user" requested enhancements, additional training, additional templates and forms is billable at $95.00 per hour plus any applicable travel and per diem (Travel cost and per diem shall be approved in advance and reimbursement will based on applicable Joint Travel Regulations (JTR).

Renewal Date: _____ Expiration Date: _____

_____

Authorized User *(signature indicates acceptance of above terms and conditions)*
(Please sign and email back to DocMaestro. sales@docmaestro.com)

Signed for and on behalf of DocMaestro:

_____

Robert Radtke, Managing Partner **Date**

DocMaestro Software Maintenance Agreement (SMA)

This Software Maintenance Agreement ("SMA") is the complete and exclusive statement of the Agreement between the original purchaser of the software and DocMaestro.

Agreement Coverage

1. Agreement is of limited duration and coverage.
2. Agreement extends only to the registered owner of DocMaestro software.
3. Agreement covers the software with any and all custom modifications incorporated in the software as delivered for the original installation.
4. Agreement provides free upgrades to the DocMaestro software as released and technical phone support during the term of this agreement. The method of delivery of any free upgrades shall be at the sole discretion of DocMaestro.
5. Any change to the install application voids this Agreement.
6. Agreement does not cover assistance in rule writing, creating rule templates, acts of God such as, but not limited to, lightning, flooding, tornado, earthquakes, and hurricanes.
7. Terms of the DocMaestro Software License Agreement are incorporated into this Agreement

Extension and Renewal of Agreement

This Agreement may be renewed on or before its anniversary date. DocMaestro may increase the SMA cost to correspond with cost of living increases, but the cost of coverage will not exceed 10% per renewal period. Any custom modifications made to the software by DocMaestro at the request of the registered owner, after the original installation, will require the issuance of an additional agreement to cover said modification.

Support Request

A Support Request web site is also available at http://www.DocMaestro.com/support. Technical support is provided by DocMaestro customer/technical support personnel who are available during the hours of 8:00 a.m. through 5:00 p.m. EST Monday through Friday, excluding regularly observed holidays. DocMaestro customer/technical support personnel can be reached via phone at (727) 443-7801 or via email at the following address: sales@docmaestro.com

Product: *InfoLinker Plus*
Serial Number: _____
Renewal Date: _____ Expiration Date: _____

_____
Authorized User *(signature indicates acceptance of above terms and conditions)*
(Please sign and email back to DocMaestro. sales@pgcsolutions.com)

Signed for and on behalf of DocMaestro, LLC:

_____
Robert Radtke, Managing Partner **Date**

Appendix B: Company Data

DocMaestro, LLC
5180 Venetian Blvd
Saint Petersburg, FL 33703
sales@docmaestro.com
http://www.DocMaestro.com