



The TimeSafe[®] Configuration Management System

Dispatch Issue Management Manual

Version 3.8

August, 2005

August 16, 2005

Dispatch Issue Management Manual

August, 2005

Copyright © AccuRev, Inc. 1995–2005
ALL RIGHTS RESERVED

TimeSafe, **AccuRev**, and **StreamBrowser** are registered trademarks of AccuRev, Inc.
Java is a trademark of Sun Microsystems, Inc.
Rational and **Rational Rose** are trademarks of IBM Corporation.

Table of Contents

Working with Issue Records	1
Accessing a Particular Issues Database	1
Creating a New Issue Record.....	1
Attaching Files to an Issue Record, 2	
Canceling Creation of a New Issue Record, 4	
Issue Records and Transactions — the Issue History Page, 4	
The server_dispatch_post Trigger, 4	
Viewing and Modifying an Existing Issue Record	5
Printing an Individual Issue Record.....	6
Using Database Queries	7
Typical Workflow, 8	
Query List Pane, 9	
Query Results Pane, 9	
The Edit Query Window: Creating and Revising Queries, 10	
Naming a New Query / Renaming an Existing Query, 11	
Creating a Simple Clause, 12	
Creating a Compound Clause, 14	
Closing the Edit Query Window, 16	
Re-executing Queries, 16	
Working with a Results Table, 16	
Selecting Columns to Appear in the Results Table, 16	
Adjusting the Widths and Order of Columns /	
Specifying a Sort Order for Rows, 17	
Working with the Records Listed in a Results Table, 18	
Additional Operations on Queries, 18	
Setting a Default Query, 19	
Deleting a Query, 19	
Saving Your Queries, 19	
Printing Query Results	19
 Designing Issues Database and Edit Forms	 21
Invoking the Schema Editor.....	21
Saving Changes to the Schema	21
Defining Database Fields	22
Integrating Configuration Management and Issue Management:	
the ‘affectedFiles’ Field and Change Packages, 23	
Data Types, 24	
Defining Multiple-Choice Lists, 24	
Designing an Edit Form	25
Form Layout Operations, 27	
Defining Edit Form Validations	28
Initializing Field Values in a New Issue Record	29
Conditional Validations	30
Specifying the Condition, 31	
Setting a Field Value, 32	
Revising the Choices for a “choose” Field, 33	
Requiring a Value to be Entered in a Field, 34	

Unconditionally Required Fields, 35	
Setting Permissions on All or Part of the Issue Record, 35	
Revising and Removing Validations and Actions, 36	
The Change Packages Tab	37

Change Packages and Integrations between Configuration Management and Issue Management39

Operations on the Versions in a Change Package	41
Replacement of Change Package Entries.....	42
Viewing Stream Contents/Differences in Terms of Change Packages	43
Formatting a Change Package Table, 43	
Promote by Change Package, 44	
Viewing a Transaction in Terms of Change Packages	44
Integrations Between Configuration Management and Issue Management	45
Change-Package-Level Integration, 46	
Enabling the Integration, 46	
Triggering the Integration, 47	
Transaction-Level Integration, 48	
Enabling the Integration, 48	
Activating the Integration, 48	
Implementation and Customization of the Transaction-Level Integration, 49	
If Both Integrations are Enabled, 49	

Working with Issue Records

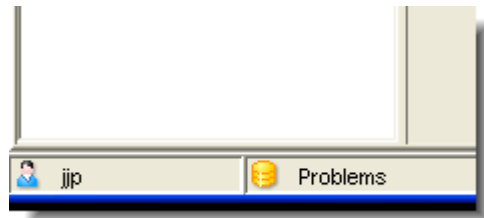
This chapter discusses usage of Dispatch, the AccuRev issue-management facility from the viewpoint of the day-to-day user. The topics include:

- *Accessing a Particular Issues Database*
- *Creating a New Issue Record*
- *Viewing and Modifying an Existing Issue Record*
- *Printing an Individual Issue Record*
- *Using Database Queries*
- *Printing Query Results*

The creation of issues databases and their edit forms is discussed in *Designing Issues Database and Edit Forms* on page 21.

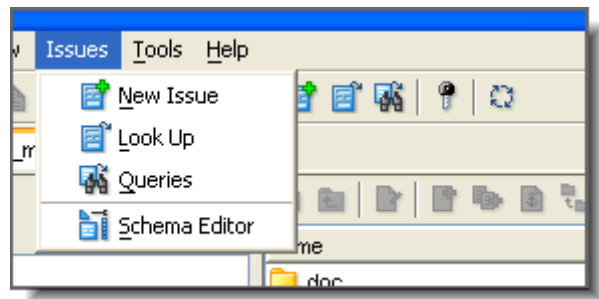
Accessing a Particular Issues Database

In the AccuRev GUI window, you can access any number of issues databases, using different tabs. The name of the depot you're using in the current tab is displayed at the bottom of the window.



When you invoke any Dispatch command from the main menu or toolbar, it applies to the current depot — the one being used by the current tab. (If there is no current depot, AccuRev prompts you to select one.)

This chapter covers **New Issue**, **Look Up**, and **Queries**. See *Designing Issues Database and Edit Forms* on page 21 for a discussion of the Schema Editor.




Creating a New Issue Record

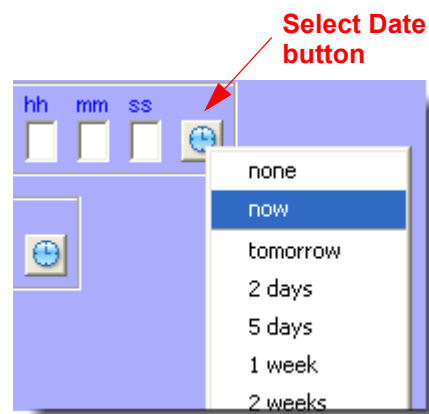


Use the **New Issue** menu command or toolbar button to open an empty edit form. The form is a new tab in the GUI's tabbed display, labeled "New Issue". Depending on the design of the issues database, you'll notice some or all of the following features:

- Certain fields have already been initialized with particular values.
- Certain field-labels are in a contrasting color (by default, red), indicating that they are required fields. You cannot save a new issue record until a value appears in all required fields. Some of those fields may have gotten their values by being initialized.

- You can't enter a value in the **Issue** field. This is the unique issue-number for this particular issue record. Dispatch fills in this field when you save the issue record.

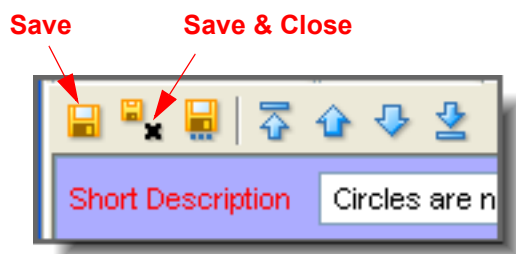
Enter values in the fields, using the mouse or **Tab** and **Shift-Tab** to move from field to field. Within a multiple-line text field, the **Tab** key inserts a TAB character, rather than jumping to the next field. Multiple-choice list-boxes work in the same way as on Web forms. Fill in the various parts of a “timestamp” field manually, or use the  **Select Date** button to fill in the fields automatically.



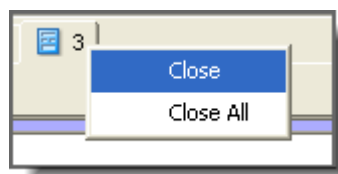
The edit form might have multiple pages, which appear as subtabs within the overall “New Issue” tab. And typically, there's a “header section”, which always remains visible as you switch from page to page. Use the mouse to switch among multiple pages.

When you're ready (have you filled in all the required fields?), click the **Save** button or the **Save & Close** button in the toolbar above the New Issue form.

Dispatch assigns an issue-number to the new record and stores it in the depot. With **Save**, the new record remains on-screen; the issue-number replaces the “New Issue” label and appears in the **Issue** field. With **Save & Close**, the tab containing the edit form disappears.



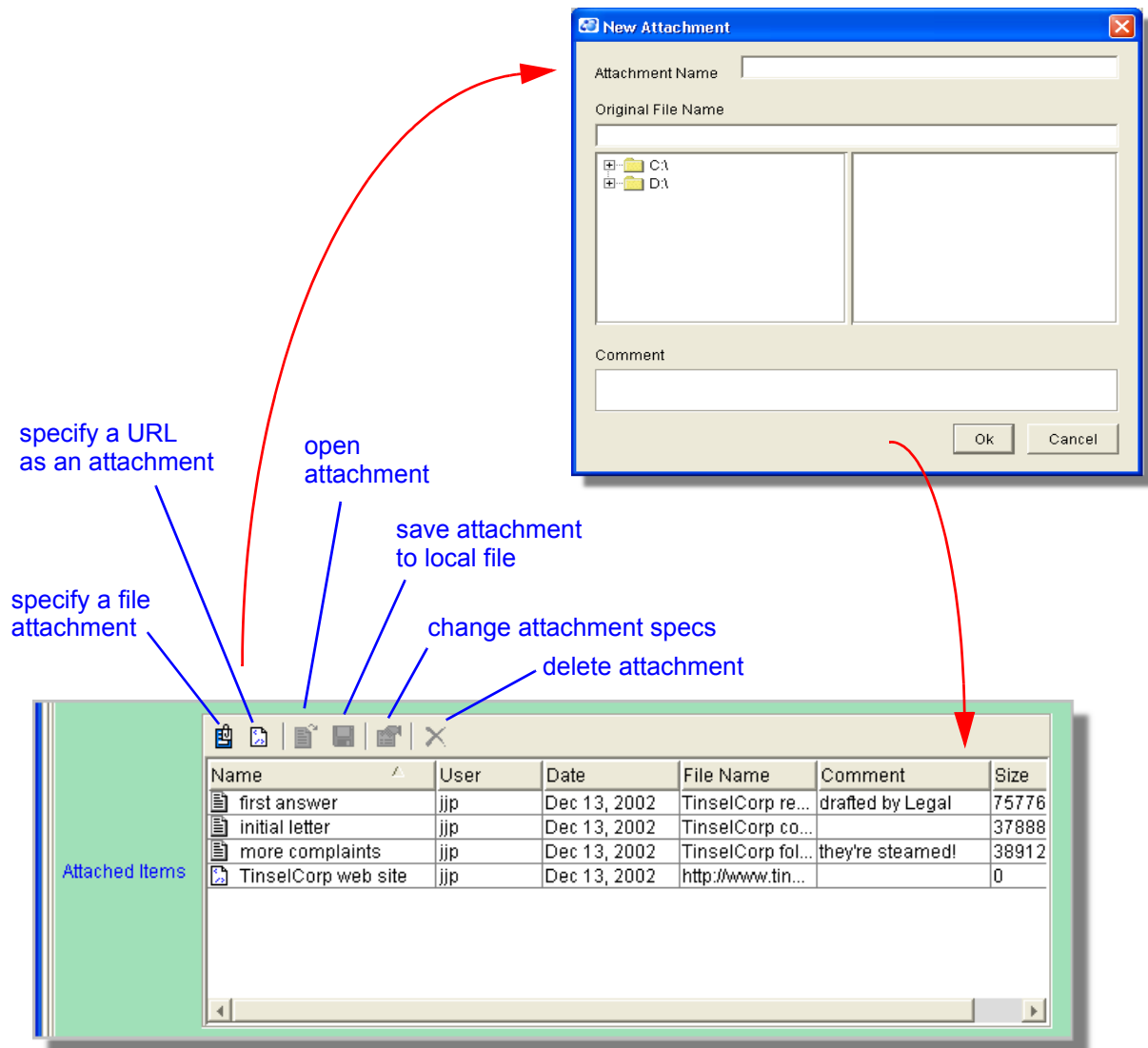
At any time, you can close an edit-form tab as you would any GUI tab: right-click the tab's title, then select **Close** from the context menu.



If you want to create another issue record, use the **New Issue** menu command or toolbar button again.

Attaching Files to an Issue Record

An edit form can contain one or more attachment fields. In each such field, you can specify one or more files and/or Internet addresses (URLs) to be attached to the current issue record. Dispatch displays the data as an attachments table.



In addition to specifying the location of a file or Internet resource, you enter a Name and optional Comment. Dispatch automatically fills in your username, the date, and the size of the attached file. (Internet URLs get assigned a size of 0.) If the edit-form field is not large enough to show all these attachment parameters, use the scroll bar to see all the data. You can also resize and rearrange the columns of an attachments table.

An attachment field includes its own toolbar, with these buttons:

New Attachment

Define a new file attachment for this issue record.

New URL

Define a new attachment to be the address (URL) of an Internet resource. Dispatch helps out by placing the string **http://** in the URL field. You can erase this if you want to specify a location accessed by another Internet protocol, such as **ftp://**.

Open Attachment

Open the existing attachment that is currently selected, using the appropriate program.

Save Attachment As

Create a copy of the currently selected attachment on the client machine.

Properties

Launch a Properties window, displaying the definition of the currently selected attachment. You can use this window to change the attachment's Name or Comment value.

Delete Attachment

Remove the attachment from the issue record (and from the depot).

When you save the issue record, each file is copied to the depot, so that the data always remains available through the issue record, even if an original file is deleted.

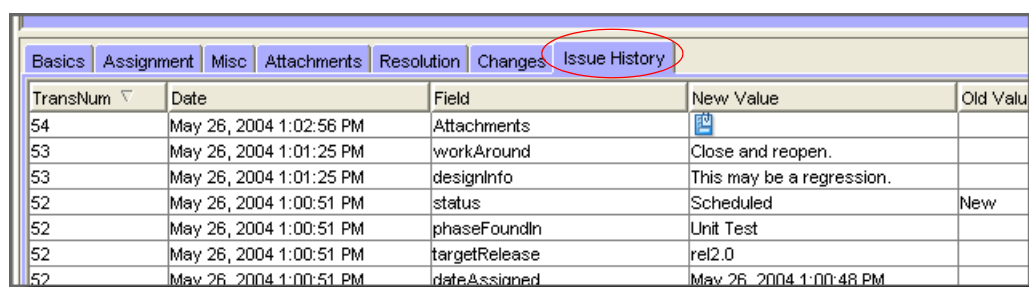
Canceling Creation of a New Issue Record

If you decide not to create a new issue record, just close the edit form (as described above) without ever clicking the **Save** button.

Issue Records and Transactions — the Issue History Page

When you create a new issue record (or update an existing record), a **dispatch** transaction is written to the depot's transaction log — the same log that records AccuRev transactions, such as **keep** and **promote**. It's important to distinguish the depot's transaction log (which is a database in its own right) from the Dispatch issues database stored within the depot.

Dispatch automatically adds an **Issue History** page to each edit form. This page shows how an issue record has



TransNum	Date	Field	New Value	Old Value
54	May 26, 2004 1:02:56 PM	Attachments		
53	May 26, 2004 1:01:25 PM	workAround	Close and reopen.	
53	May 26, 2004 1:01:25 PM	designInfo	This may be a regression.	
52	May 26, 2004 1:00:51 PM	status	Scheduled	New
52	May 26, 2004 1:00:51 PM	phaseFoundIn	Unit Test	
52	May 26, 2004 1:00:51 PM	targetRelease	rel2.0	
52	May 26, 2004 1:00:51 PM	dateAssigned	May 26, 2004 1:00:48 PM	

changed over time. It displays all the **dispatch** transactions for an issue record, except for the initial **Save**. Each row of the table shows the change to one field made by a subsequent **Save**.

If you double-click any row of the Issue History table, Dispatch opens a read-only edit form, showing the state of the issue record at the time of that particular **Save**.

The server_dispatch_post Trigger

After Dispatch saves a new issue record (or updates an existing record), it looks in the AccuRev executables (**bin**) directory on the server machine for a script named **server_dispatch_post**. If

such a script exists, it is executed on the server machine, under the identity of the user account that runs the **accurev_server** process. This trigger is intended to enable email notification of interested parties — for example, the user to whom a bug has been assigned. A sample Perl script is available in the **examples/dispatch** subdirectory of the AccuRev installation directory.

On Windows machines, the actual script name must be **server_dispatch_post.bat**. As with other triggers, you must convert a Perl script to a batch file, using the conversion utility **pl2bat**.

On Unix machines, the name can be **server_dispatch_post** (an executable binary or shell script) or **server_dispatch_post.pl** (an executable Perl script).

Note: for compatibility with previous AccuRev releases, the script can also be named **dispatch_email**, with the appropriate suffix.

Viewing and Modifying an Existing Issue Record

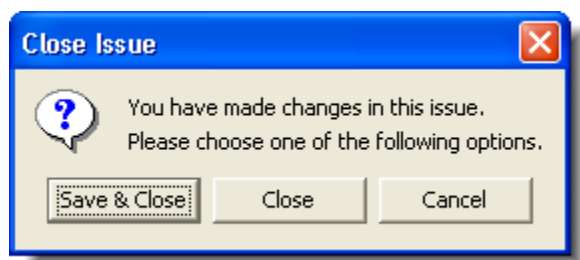
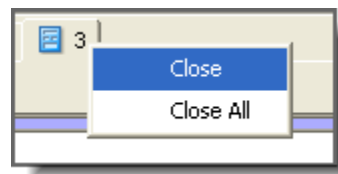


The easiest way to retrieve an existing issue record is through its unique issue-number. Use the **Look Up** menu command or the **Open Issue** toolbar button. Dispatch prompts you to enter an issue-number, retrieves the record, and displays it in the edit form.

Note: issue-numbers are unique within a given depot. Several different depots might have an issue-record numbered 382. Make sure you've opened the right depot!

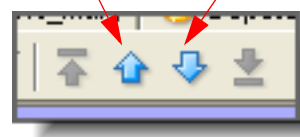
If you wish, modify one or more fields, then click the **Save** or **Save & Close** button. Required-fields restrictions apply when you're modifying an existing issue record, just as they do when you're creating a new one.

If you don't want to modify the issue record, or if you change your mind after modifying some fields, just close the GUI tab containing the edit form without clicking the **Save** button. Dispatch asks for confirmation, to make sure that you don't mistakenly discard work that should be saved:



While you're viewing or modifying an issue record that you've displayed with **Look Up**, browse arrows are enabled in the edit form's toolbar. This makes it easy to view a set of consecutive issue records. If you've made some changes to an issue, Dispatch prompts you to save or discard those changes before switching to the previous or next one.


Previous record **Next record**

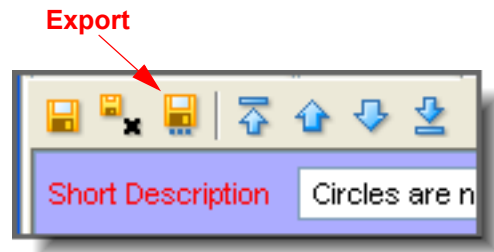


As described in section *Issue Records and Transactions — the Issue History Page* on page 4, the updating of an issue record is recorded as a **dispatch** transaction in the depot's transaction log.

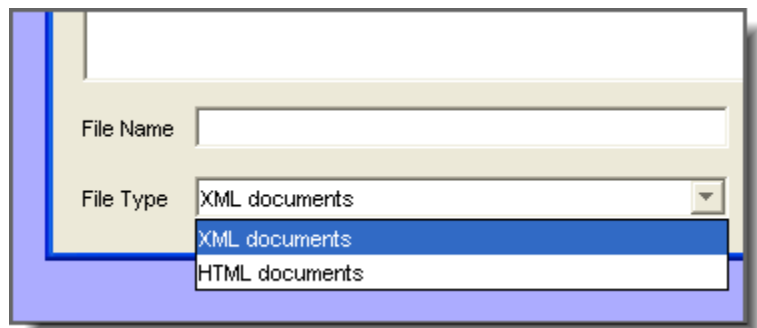
Printing an Individual Issue Record

At any time when you're using an edit form, whether creating a new issue record or modifying an existing one, you can “print” it. For Dispatch, printing means “publish to the Web” — which means either “create an HTML file” or “create an XML file”.

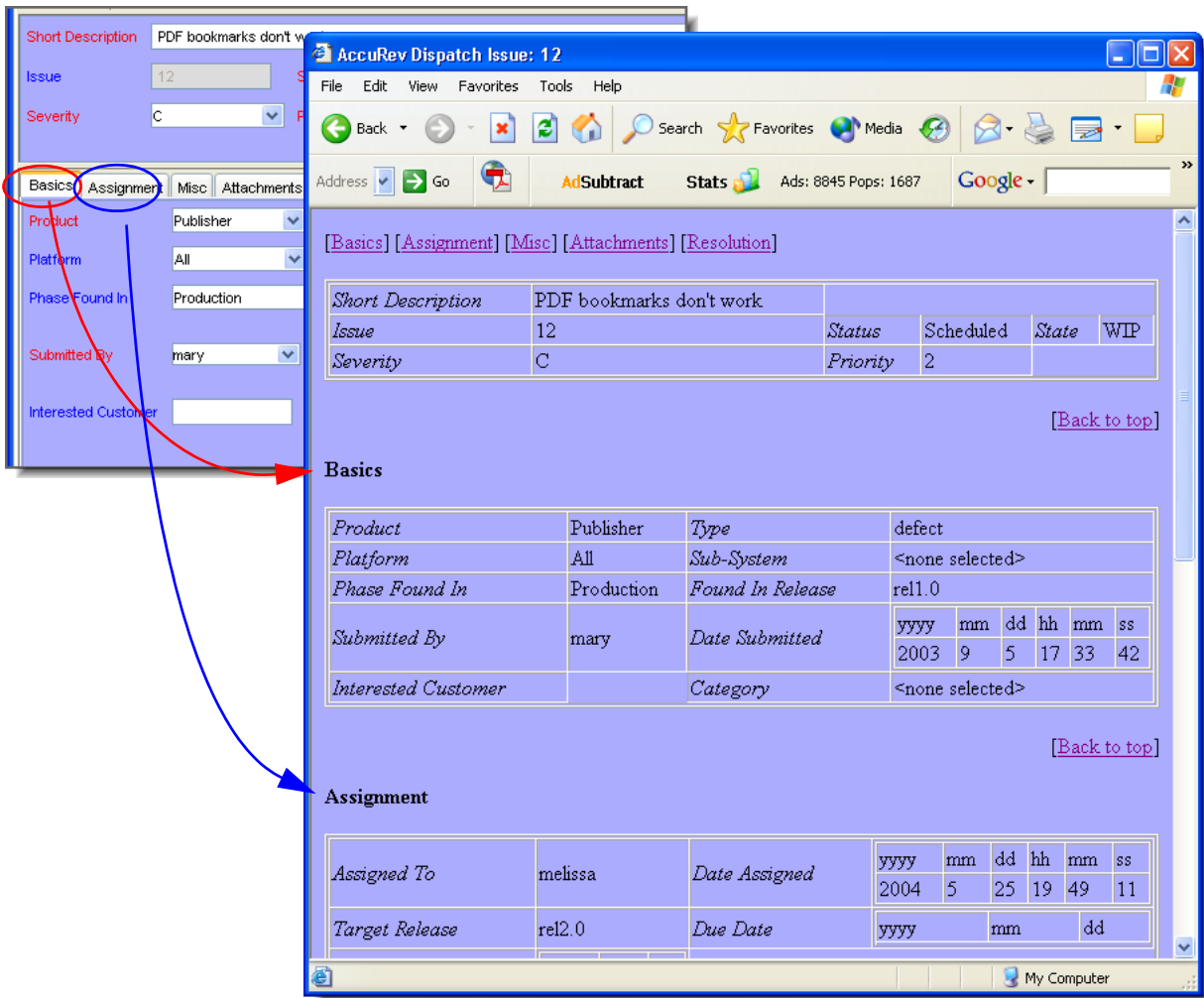
- Click the  **Export** button in the form's toolbar.



- Select the file type in the File Chooser window that appears.
- Specify a pathname for the export file, using the navigator and the File Name input field. (You don't need to specify the **.html** or **.xml** suffix — Dispatch adds it automatically.)



When creating an HTML file, Dispatch outputs all the content of the issue record and approximates the form layout, too. Even if the edit form has multiple pages, you need to “print” only once. All of the pages are combined into a single HTML or XML file:



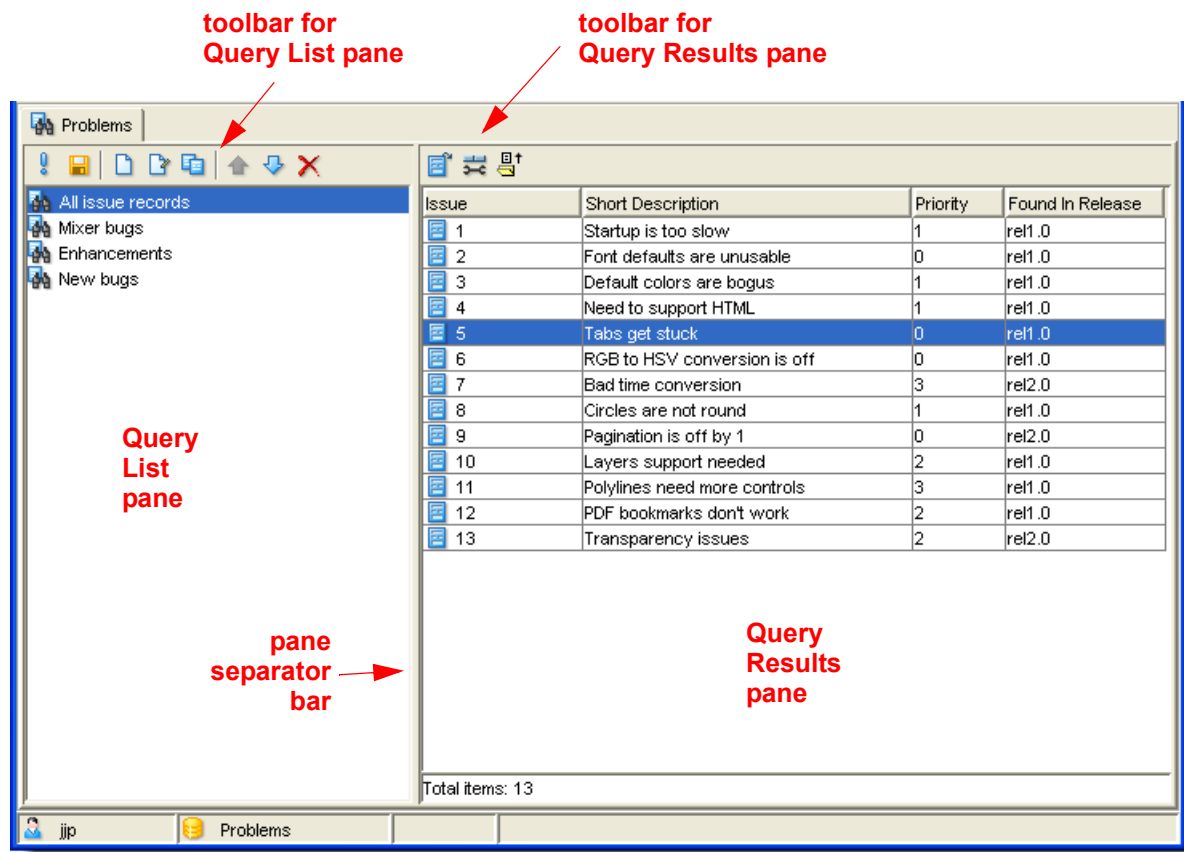
On Windows machines, Dispatch automatically invokes a Web browser on the HTML file it creates. If you wish, resize the browser window to optimize the look of the issue record. HTML documents automatically adjust to changes in window width. Then, use the browser's print command to create hardcopy of the issue record.

Using Database Queries

Retrieving one issue record at a time is useful in many situations. But robust issue-management systems must also support queries, which retrieve groups of records according to user-defined selection criteria. Dispatch has a point-and-click interface for creating and editing queries; it enables you to create simple queries quickly, and to create sophisticated queries in a straightforward, reliable way.



Use the **Issues > Queries** command or the **Queries** toolbar button to open a new Queries tab in the GUI window. The Queries tab includes two panes, each with its own toolbar:







- The **Query List** pane lists the names of all your existing queries. (Each AccuRev user has his own set of queries; there are no “global” queries available to all users.) Using the toolbar in this pane, you compose new queries, view/revise/rearrange/execute existing queries, and delete queries.
- When you execute a query, the set of issue records selected by the query are displayed as a results table in the **Query Results** pane. If you’ve set a default query, it’s executed automatically when you open the Queries tab. See *Setting a Default Query* on page 19. Dispatch remembers query results as long as the Queries tab stays open. It’s easy to browse through the results of several queries, switching back and forth instantly between different queries’ results tables.

You can use the separator bar between the panes to adjust their relative size. And remember that the GUI window itself is resizable, too.

Typical Workflow

Here’s a typical workflow for creating and using a query:



1. Create a new query: click the  **New Query** button to open the Edit Query window with a blank query, assign the query a name, and specify one or more clauses that select issue records.

2. In the Query Results pane, select click the  **Setup Columns** button and select certain fields to appear in the results table.
3. Click the  **Run Query** button to execute the query, loading data from selected issue records into the results table.
4. Optional: save the results table in HTML format; use a Web browser to view and print the table.
5. Use the  **Save All** button on the Queries toolbar to save the query in the depot, for future use. Each user's queries are stored separately.

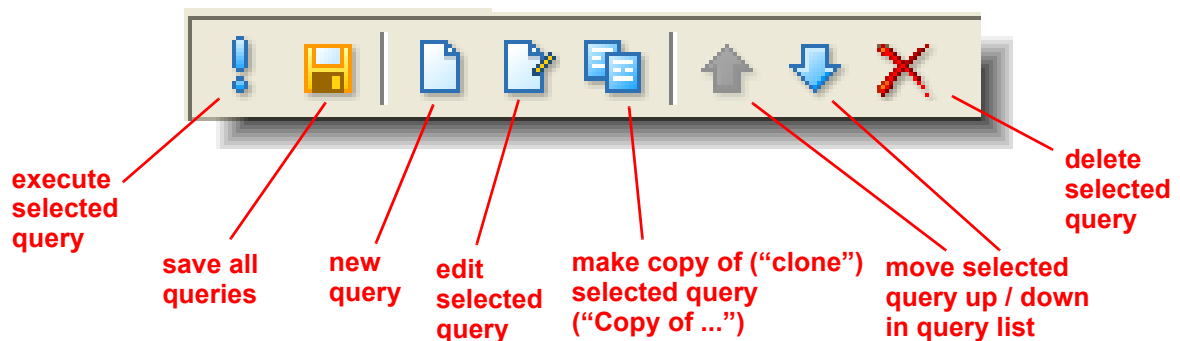
You can refine a query as much as you want, changing the record-selection criteria and the fields that appear in the results table.

The following sections describe the components of the Queries tab in more detail.

Query List Pane

The Query List pane contains a list of the queries you've defined for the current depot (issues database). The listing is not alphabetical — whenever you create a new query, it's simply added to the end of the list. You can rearrange the order of the queries using the toolbar buttons  and .

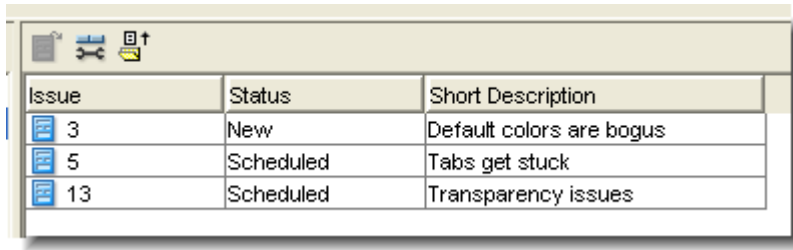
The Query List pane has its own toolbar:



These commands are also available on the context menu of individual queries in the Query Editor pane. The context menu also contains the command **Set as Default**, which designates the query to be the default query for the current issues database. See [Setting a Default Query](#) on page 19. If a query is already the default, the context menu item becomes **Disable as Default**.

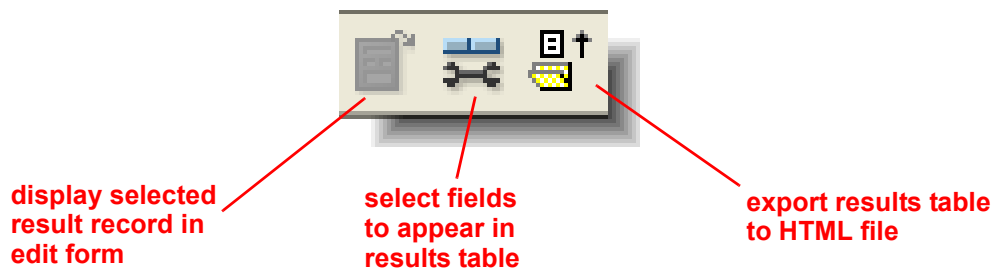
Query Results Pane

The Query Results pane displays the results of a query as a results table. Each row of the table displays one issue record; each column displays a particular issue-record field.

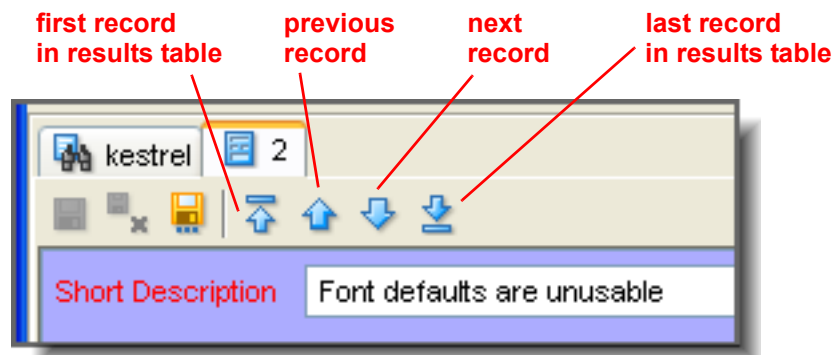


Issue	Status	Short Description
3	New	Default colors are bogus
5	Scheduled	Tabs get stuck
13	Scheduled	Transparency issues

Using the mouse, you can rearrange and resize the table's columns. You can also specify a single-level or multiple-level sort order for the table's rows. Additional operations are available through the Query Results pane's toolbar:





- **Open Issue:** Open an edit form to view/revise the selected issue record in the results table. In that edit form, toolbar buttons enable you to browse some or all of the other records in that result table.



- **Setup Columns:** Change which columns (fields) appear and their order
- **Export Table:** Create an HTML file containing the entire contents of the results table.

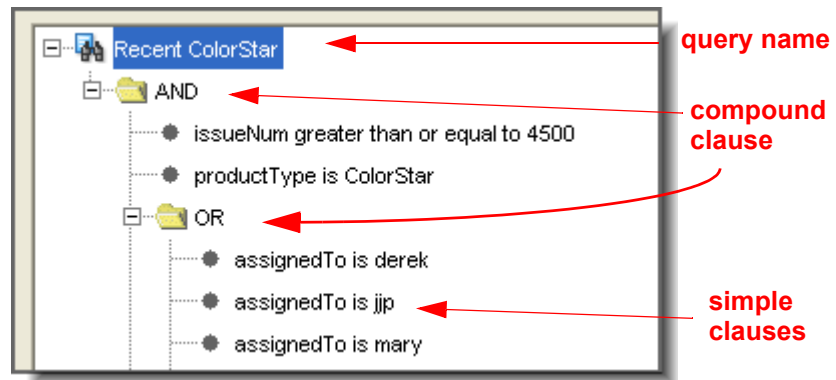
The commands (except for Export Table) are also available on the context menu of individual queries in the Query Editor pane. An issue record's context menu also contains the **Remove Column** command, which provides a quick way to revise the structure of the results table. (Dispatch knows which column you've right-clicked on to display the context menu.)

The Edit Query Window: Creating and Revising Queries

The Query List pane's toolbar includes  **New Query** and  **Edit Query** command buttons. These commands open an Edit Query window, in which you compose a new query or revise an

existing one. A query is displayed as a hierarchy, which you navigate using the familiar expand/collapse (+/–) controls. The hierarchical organization is a natural fit, because each query is, itself, a hierarchy of simple clauses and compound clauses:

- The first level contains the query’s name.
- A simple clause, such as “value of the **assignedTo** field is **mary**”, can live at any level.
- A compound AND clause (*cls-a* AND *cls-b* AND *cls-c* AND ...) consists of the operator AND at one level of the hierarchy and any number of clauses *cls-a*, *cls-b*, *cls-c*, etc. at the next sublevel.
- Compound OR clauses are structured hierarchically in exactly the same way.




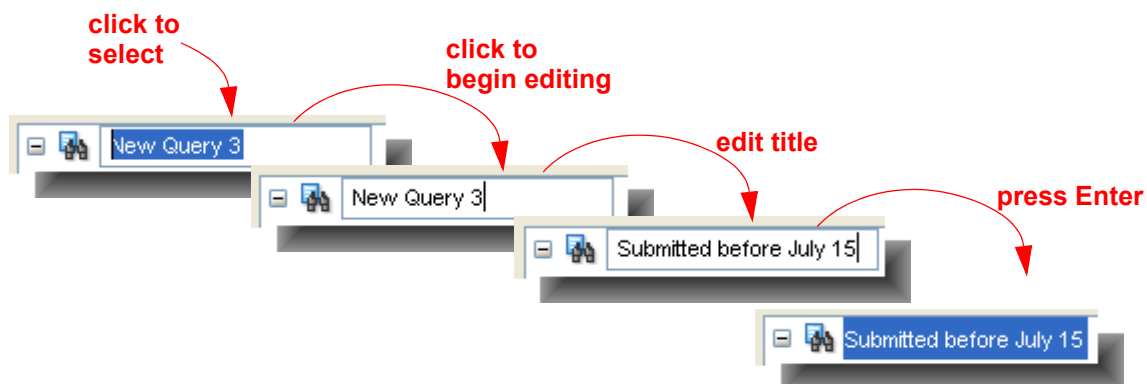
The hierarchy can get deep, because each of the clauses *cls-a*, *cls-b*, etc. can be either simple or compound. For example, the sophisticated query pictured above can be expressed as:

“Retrieve each bug report that is assigned to either **derek**, **jjp**, or **mary**, and applies to product **ColorStar**, and is numbered above 4500.”

The following sections fill in the details of working in the Edit Query window.

Naming a New Query / Renaming an Existing Query

When you click the  **New Query** button to create a new, empty query, Dispatch assigns it a placeholder title (“New Query *nnn*”). You can edit the title now or whenever the Edit Query window is open: click the title once to select it (if it is not already selected); then click a second time to begin editing it. Not too fast! Double-clicking a title is equivalent to using the expand/collapse control.

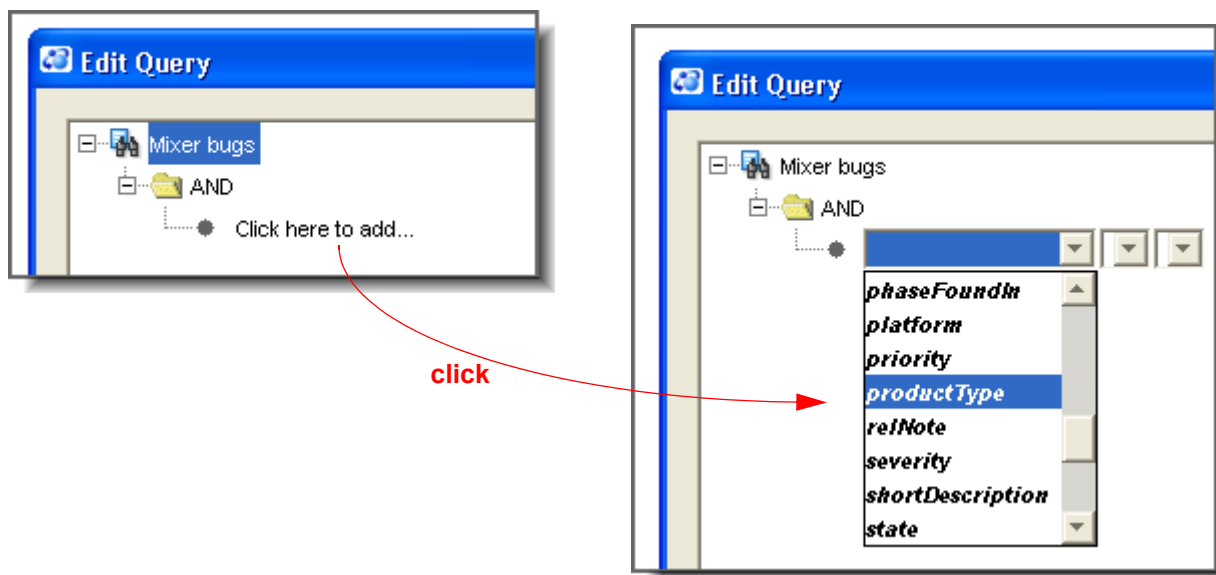


Creating a Simple Clause

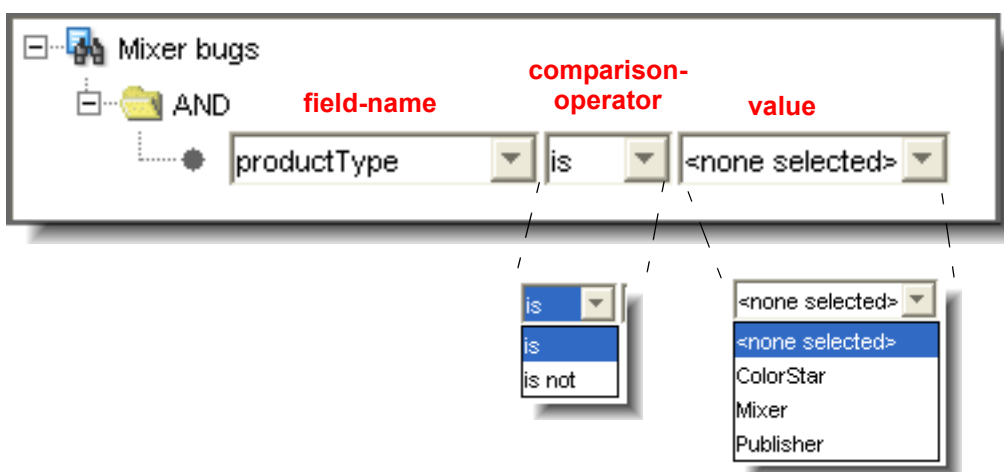
Every query consists at least one simple clause. A simple clause has three parts:

<field-name> *<comparison-operator>* *<value>*

The point-and-click interface makes creating a simple clause easy and (almost) foolproof. Start by clicking one of the “Click here to add ...” placeholders in the query:



First, you must select the *<field-name>* part of the clause from the list-box containing all the field-names. When you select a field-name, the query editor automatically adjusts the *<comparison-operator>* and *<value>* parts of the clause, based on the selected field. In the example below, the user has selected field-name **productType**, whose value must be one of these names: **ColorStar**, **Mixer**, **Publisher**.



The table below shows all the Dispatch data types, along with the corresponding choices for the *<comparison-operator>* and *<value>* parts of a simple clause. For more on data types, see [Data Types](#) on page 24.

Field Type	Comparison Operators	Values
Text	matches contains equal to not equal to less than less than or equal to greater than greater than or equal to	<p>Any character string. (Do not enclose it in quotes.) The value is always interpreted as a string literal; there is no way to specify the value of some other field here.</p> <p>The comparison is always a string comparison, never a numeric comparison. For example, the value 3 is greater than the value 25.</p> <p>The matches and contains operators are case-insensitive.</p>
Timespan	equal to not equal to less than less than or equal to greater than greater than or equal to	<p>A numeric value, representing an amount of time.</p> <p>Note: users specify the value in the edit form as a number of <i>hours</i> (e.g. 7.5); an XML-format dump of the issue record created by the Export command reports the value as a number of <i>minutes</i> (e.g. 450).</p>
Choose	is is not	One of the strings specified in the definition of this field in the Schema Editor.
List	is is not	One of the strings specified in the definition of a particular named list in the Schema Editor.
User	is is not is member of is not member of	One of the principal-names in the user registry maintained by the AccuRev server. Alternatively, a user-group defined in the registry.
Timestamp	is is not is before is after is before or equal to is after or equal to	An AccuRev timestamp.
Attachments	contains	Any character string. This string is compared to the Name of each of an issue record's attachments. See Attaching Files to an Issue Record on page 2.
internal	equal to not equal to less than less than or equal to greater than greater than or equal to	An integer, identifying a particular Dispatch issue record (issueNum field) or a particular AccuRev transaction (transNum field).

As you “fill in the blanks” to create simple clauses, you’ll notice that Dispatch allocates new “Click here to add ...” placeholders. It makes sure that one of these placeholders is always available at each level of the query.

Creating a Compound Clause

A compound clause combines any number of subclauses together, using the same logical operator: AND or OR. (The NOT operator is not supported.) The subclauses to be combined can, themselves, be either simple or compound:

simple AND simple
simple AND compound
simple OR compound OR simple
compound AND compound AND simple AND compound AND compound
etc.

Note: a compound clause can contain a single subclause. This is logically equivalent to using the subclause by itself. In fact, the most basic query you can create is a compound AND clause (or compound OR clause) with one simple subclause.

What about this?

simple OR simple AND simple

Well, it depends. This might be a compound OR clause within a compound AND clause:

((simple OR simple) AND simple)

Or it might be a compound AND clause within a compound OR clause:

(simple OR (simple AND simple))

Whichever meaning was intended, Dispatch models the logical statement as one compound clause nested within another.

We saw above that the query editor automatically creates placeholders for simple clauses. But you must explicitly insert a compound-clause placeholder yourself, then fill in the subclauses. We’ll demonstrate with an example:

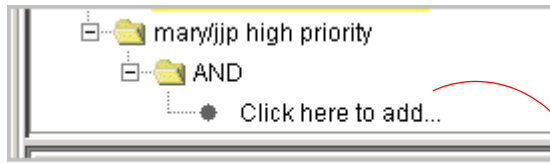
“Retrieve all **high**-priority bug reports assigned to either **mary** or **jjp**”

... which becomes a compound OR clause within a compound AND clause:

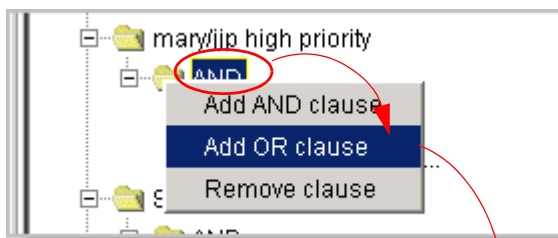
*(fix_priority is **high** AND (assign_user is **mary** OR assign_user is **jjp**))*

Here’s the procedure for creating this query:

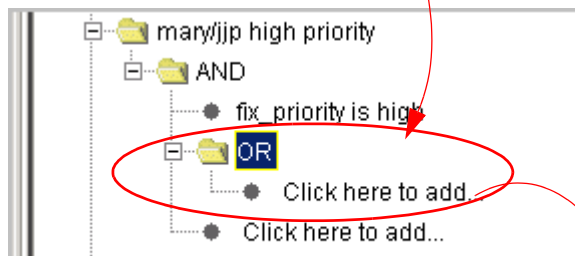
create a new query and name it



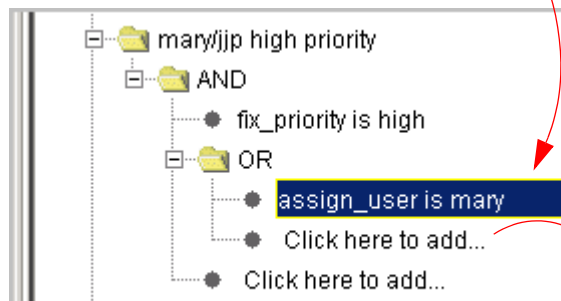
click the placeholder and fill in a simple clause



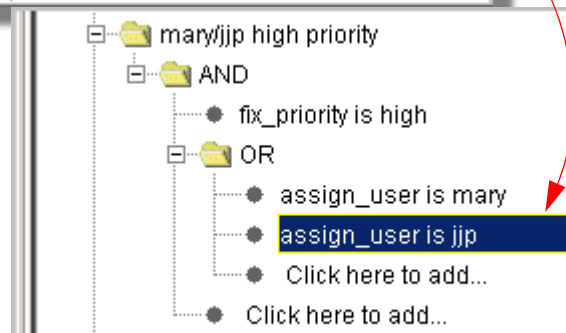
right-click at the parent level and create an OR subclause



click the placeholder and fill in a simple clause

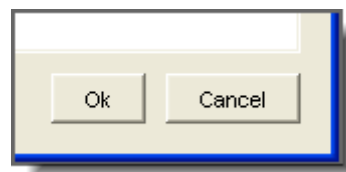


click the placeholder and fill in a simple clause




Closing the Edit Query Window

Whether you're composing a new query or revising an existing one, you end by saving your work (**Ok** button) or discarding it (**Cancel** button). Dispatch automatically executes the query and displays a results table in the Query Results pane.



At this point, you'll often want to work on the design of the results table: specify additional fields to be displayed, and adjust the widths and order of the columns. For more on this, see *Working with a Results Table* on page 16.

Re-executing Queries

 Whenever you select a particular query in the Query List pane, Dispatch displays in the Query Results pane the most recent results of running that query. (The cache of previous query results is cleared when you close the Queries tab.) You can update a results table to reflect recent Dispatch transactions by clicking the **Run Query** button on the Query List toolbar.


You don't need to click the **Run Query** button when you revise a query in the Edit Query window — Dispatch automatically executes the revised query and updates the results table.

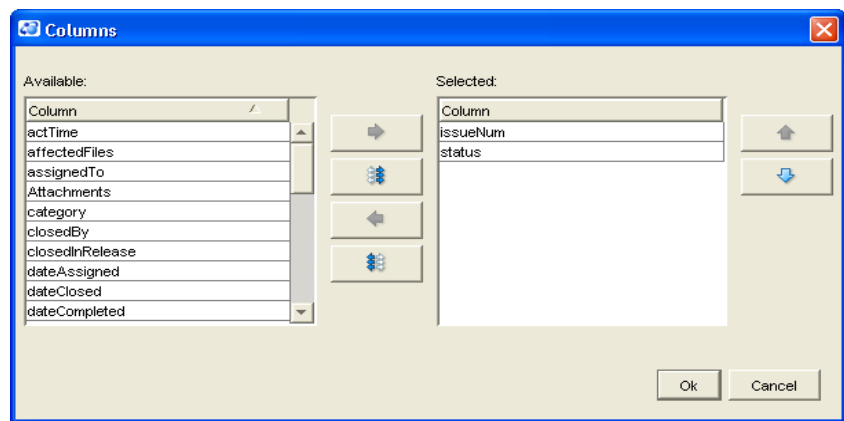
Working with a Results Table

Each query has its own results-table design: a set of columns (fields), in a particular order and with particular column-widths. The results table's design can also include a single-level or multiple-level sort order for the rows (issue records).

When you create a new query, Dispatch automatically starts off the results table with a couple of columns, including **Issue** (issue-number). You can add more columns at any time. You can also remove the **Issue** column from the table.

Selecting Columns to Appear in the Results Table







 Use the **Columns** button in the Query Results toolbar to launch a window in which you select the columns to appear in the results table, and their order. (You can also change the column order in a results table using drag-and-drop — see below.)



Note that:

- The names that you work with in the Columns window are the official field-names defined in the issues database schema.
- The names that appear as column headers in the results table are the labels that appear on the database's edit form.

For some queries, you may want to include just a few important columns in the results table; for others (e.g. a complete data-dump), you may want to include all the fields defined in the issues database. Working in the Columns window, you can:

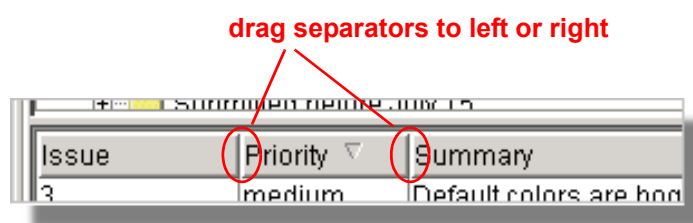
- Select a column label and use the  and  buttons to move it between the **Selected** (appears in results table) and **Available** (does not appear in results table) lists.
- Select a column label in the **Available** list and use the  and  buttons to change the order of the columns.
- Use the  and  buttons to move all columns labels to the **Available** or **Selected** list.

When you're done, click **Ok** to apply the changes you've made, or click **Cancel** to discard the changes.

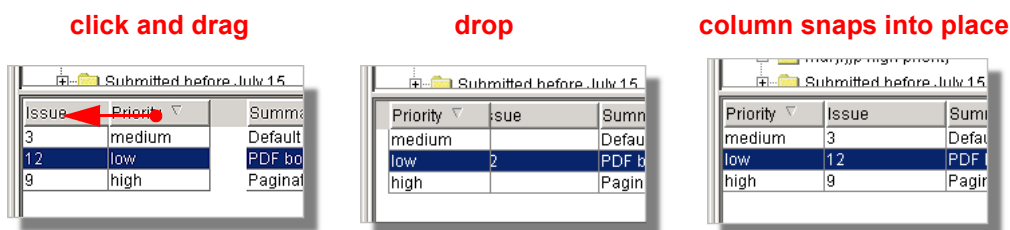
Adjusting the Widths and Order of Columns / Specifying a Sort Order for Rows

In the Query Results pane, you can adjust columns widths, change the column order (overriding any changes you made in the Columns window), and/or specify a sort order for the rows:

- Drag column separators to adjust the column widths:



- Drag column headers to rearrange columns:



- Sort the results-table rows, on a single column or on multiple columns. The sorting techniques are the same as in the File Browser's details pane.

All the changes you make, both in the Columns window and in the Query Results pane, are preserved when you invoke the **Save All** command. See *Saving Your Queries* on page 19.

It's likely that you won't be able to see the complete text strings entered into multiple-line text fields, no matter how wide you've made the results-table

Priority	Problem
high	The "Repaginate" command always produces a document that
low	The default font should be at least 10 pts. It should be user-con
	Conversion from 24-hour format to AM/PM is backwards

line-terminator character

column. This limitation doesn't apply when you "print" the results table to an HTML file: the complete contents of each field is included in the HTML table. (See *Printing Query Results* on page 19.)

If you change the column layout in the Query Results pane after running a query, you don't need to rerun it — even if you add or delete columns. Dispatch automatically updates the results table in this case.

Working with the Records Listed in a Results Table

In many cases, browsing the results table produced by running a query may be all you need to do. But in other cases, you may want to see a selected issue record in the context of its edit form:

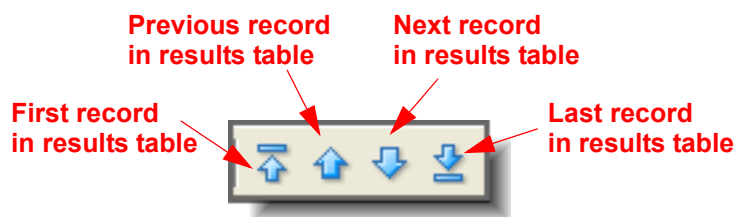
- The results table is read-only. To modify an issue record, you must access it through its edit form.
- That data you need may be in a field that is not included in the results table. Instead of adding a column to the results table, you can use the edit form to display all the fields.



To retrieve one of the records listed in the results table, select that row of the table and click the **Open Issue** button in the Query Results toolbar. Alternatives: right-click the results-table row and select **Open Issue** from the context menu; or simply double-click the row in the results table.

If you wish, modify the record as described in *Viewing and Modifying an Existing Issue Record* on page 5. After modifying a record, you may want to rerun the query in order to update the results table; this doesn't occur automatically.

When you open an issue record from a results table, browse arrows are enabled in the edit form's toolbar. This makes it easy to view, in the edit form, some or all of the records selected by a query.



Additional Operations on Queries

The following sections describe additional operations on individual queries and their results tables.

Setting a Default Query

You can designate one of your queries to be the default query for this issues database. Right-click the query in the Query List pane and select **Set as Default** from the context menu. The query name is redisplayed in ***bold-italic*** to indicate that it's the default query.

The default query is executed whenever you open a new Queries tab.

Note: prior to Version 3.5, the query was also executed whenever the GUI's **Promote** command was executed, as part of the integration between Dispatch and AccuRev (if the integration was enabled). As of Version 3.5, there's a specially defined query to support the integration. See *Viewing Stream Contents/Differences in Terms of Change Packages* on page 43. If no specially defined query exists, AccuRev reverts to executing the issues database's default query.

A query loses its status as the default query when you select **Disable as Default** from its context menu, or when you select another query as the default.

Deleting a Query



Use the **Delete Query** button to remove the currently selected query. The deletion does not take effect until you save your queries (see *Saving Your Queries* below). If you close the Queries tab without saving your queries, the “deleted” query will still exist the next time you open the Queries tab. This may or may not be the right thing to do: closing the Queries tab without saving also discards changes you've made to other queries since the most recent save.

Saving Your Queries



Use the **Save All** button on the Queries toolbar to save all the queries in the Query Editor pane. (There is no way to save a single query.) Your queries are stored within the depot directory, in an XML-format file:

```
.../storage/<depot-name>/dispatch/config/user/<principal-name>/query.xml
```

Note the *<principal-name>* directory in this pathname; the queries for each user are stored separately.

Although your queries are stored in the depot, they are not version-controlled in the way AccuRev files are. For example, there is no command that displays or reinstates your queries as you saved them two days ago.

Printing Query Results




Use the **Export Table** button on the Queries toolbar to save the current contents of the results table to an HTML-format file. The file contains an HTML table; each cell changes height and width when you view it with a Web browser and adjust the size of the browser window. This is particularly useful for viewing the contents of multiple-line text fields.

Issue	Priority	Summary	Program	Re
2	low	Font defaults are usable	ColorStar	2.0
3		Default colors are bogus	Mixer	2.0
4	high	Need to support HTML	Publisher	2.0
5	high	Tabs get stuck	Mixer	2.0
6	high	RGB to HSV conversion is off	ColorStar	2.0
7	medium	Bad time conversion	Publisher	1.1
9		Pagination is off by one	Publisher	2.0
10	high	Layers support needed	Mixer	3.0
11	low	Polylines need more controls	ColorStar	2.0
12		PDF bookmarks don't work	Publisher	2.0

Back Forward Stop Refresh Home

Address [D:\403.html](#) Go

Issue	Priority	Summary	Program	Release
2	low	Font defaults are usable	ColorStar	2.0.1
3		Default colors are bogus	Mixer	2.0.1
4	high	Need to support HTML	Publisher	2.0
5	high	Tabs get stuck	Mixer	2.0.1
6	high	RGB to HSV conversion is off	ColorStar	2.0.2
7	medium	Bad time	Publisher	1.1



“print” to HTML file

HTML file, displayed in web browser

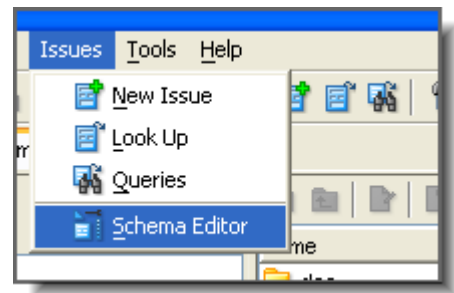
Designing Issues Database and Edit Forms

This chapter describes how to define a Dispatch issues database, containing a collection of issue records. Each depot can contain one issues database, along with a custom-designed edit form, through which users create and modify the issue records. You can make the edit form “smart” by defining validations (edit checks) that specify default values, required fields, and interrelationships among multiple fields.

Invoking the Schema Editor

You perform all Dispatch database-design work on the Schema Editor tab, which you display using the **Issues > Schema Editor** command.

The first time you invoke this command in a particular depot, Dispatch offers to use the repository’s default schema. Accepting this offer copies a set of XML-format configuration files from subdirectory **dispatch/config** of the **site_slice** directory to this depot.



Note: the default schema does not actually become the schema for this depot until you click the Schema Editor’s **Save** button. See *Defining Database Fields* below.

The Schema Editor tab includes these subtabs:

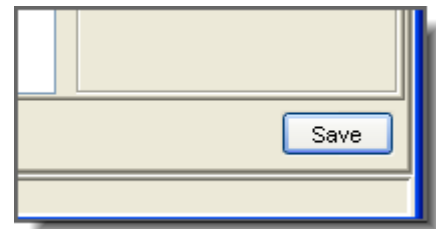
- You define the Dispatch database’s fields on the Schema subtab.
- You design the database’s edit form on the Layout subtab.
- You define multiple-choice lists, each of which can constrain the values of one or more fields, on the Lists subtab.
- You define field validations, or edit checks, on the Validation subtab.
- (As of Version 3.5) You control certain aspects of the integration between AccuRev and Dispatch on the **Change Packages** subtab.



Saving Changes to the Schema

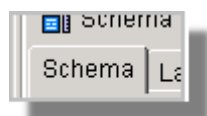
At any time while working in the Schema Editor, you can click the **Save** button in the lower right corner of the Schema Editor tab. This saves the current state of the schema to a set of XML-format files in subdirectory **dispatch/config** of the depot directory (slice) in the AccuRev repository:

- Contents of the **Schema** subtab: **schema.xml**
- Contents of the **Layout** subtab: **layout.xml**



- Contents of the **Lists** subtab: **lists.xml**
- Contents of the **Validation** subtab: **logic.xml**
- Contents of the Change Package Results section of the **Change Packages** subtab: **cpk_fields.xml**
- Contents of the Change Package Promote Triggers section of the **Change Packages** subtab: **cpk_promote_queries.xml**

Defining Database Fields



To begin designing an issues database, open a Schema Editor tab and go to the Schema subtab. (This happens automatically if you accept Dispatch's offer to use the repository's default schema.) If you are not using the default schema, Dispatch initializes the Schema Fields table with two fields.

- **issueNum**: An integer-valued field that records the position of the issue record in the depot's issues database. This number is assigned when a user creates the record (i.e. at the first **Save** on the edit form), and it never changes.
- **transNum**: An integer-valued field that records the transaction number of the most recent update to the issue record. The transaction resides in the depot's transaction log — the same log that records AccuRev transactions, such as **keep** and **promote**. The type of the issue-record-update transaction is **dispatch**.

Note that a record's **issueNum** value never changes, but its **transNum** value changes each time a user **Saves** the record. The integration between AccuRev and Dispatch also updates issue records, and so changes the value of the **transNum** field. Also note that **issueNum** and **transNum** are indexes into two different databases.

You can define any number of additional fields in the issues database. Follow these steps for each new field:

1. Click the **Add** button at the bottom of the Schema subtab.
2. Fill in the **Create New Field** window that appears:

- **Name**: The official field-name of the field. Users of the Dispatch database don't ever need to know this name — they know the field by its label. The field-name must not contain any SPACE characters. Validations must be expressed in terms of the field's name, not its label.
- **Type**: One of the data types supported by Dispatch. See *Data Types* below.

 A screenshot of the 'Create New Field' dialog box. It has a title bar with a blue background and a red close button. The dialog contains four input fields: 'Name' with the text 'problem_headline', 'Type' with a dropdown menu showing 'Text', 'Label' with the text 'Summary', and 'Report Width' with the text '30'. At the bottom right are 'Ok' and 'Cancel' buttons.

- **Label:** The field-label character string that identifies the field on the database's edit form. A field-label can contain SPACE characters (e.g. **Last Name**).
 - **Report Width:** An integer that determines the relative width of the field in the HTML table created when the user clicks **Export** on the edit form of an individual issue record.
3. Click **OK** to close the **Create New Field** window.
 4. In the Field Values box to the right of the Schema Fields table, specify additional information about the field. The kind of information required varies with the data type (see *Data Types* below).

issueNum	internal	Issue	10
problem_description	Text	Problem	30
problem_headline	Text	Summary	30
program_name	List	Program	15
program_release	List	Release	6
state	Choose	State	10
submit_date	Timesta...	Submitted on	15
submit_user	User	Submitted by	10

Height:

Width:

Repeat the steps above as often as required to create new fields in the Dispatch database.

Note: your field definitions are not saved until you click the **Save** button in the lower-right corner of the Schema Editor tab. You cannot save your work until you place at least one field in the database's edit form (Layout subtab).

Integrating Configuration Management and Issue Management: the 'affectedFiles' Field and Change Packages

If you wish to enable the integration of a depot's version-controlled files and its Dispatch issue records, define a database field whose name is **affectedFiles**. The field's type must be "text"; see *Data Types* below. You can choose any label for the field. (Such a definition is included in the default Dispatch database schema.)

The integration also depends on the enabling of a built-in AccuRev trigger procedure:

```
accurev mktrig -p <depot-name> pre-promote-trig client_dispatch_promote
```

The integration routine writes the transaction number of each **promote** command to the **affectedFiles** field of a particular issue record. Alternatively (as of Version 3.5, AccuRev Enterprise only), the integration routine records each promoted version in the issue record, in a special section named **Changes**. This section is maintained automatically by Dispatch — you don't need to define any database fields to enable this additional aspect of the integration.

For details, see *Integrations with Dispatch* on page 133 of the *AccuRev User's Manual (CLI)*, and *Change Packages and Integrations between Configuration Management and Issue Management* on page 39.

Data Types

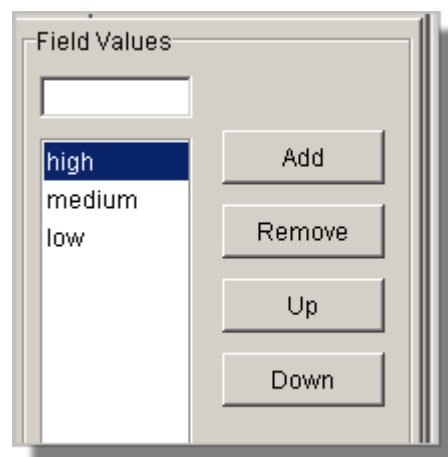
Each field you define in the **Create New Field** window must have one of the Dispatch data types listed in the table below.

Field Type	Possible Values	Required Additional Information in “Field Values” Box
text	Any character string. For multiple-line fields, the string can include line-terminators.	Height: number of lines displayed for the field in the edit form (default: 1). For multiple-line fields (height > 1), the edit form includes an expand/contract button to increase/decrease the number of lines displayed. Width: relative width of field in the edit form.
timespan	A numeric value, indicating a number of hours. Decimal values (e.g. 4.5) are allowed.	None.
choose	One of the strings specified in the Field Values box for this field.	A set of strings. In the edit form, a list-box containing this set of strings is offered to the user.
list	One of the strings specified in the definition of a particular named list.	The name of an existing list, defined on the Lists subtab. (See <i>Defining Multiple-Choice Lists</i> below.) In the edit form, a list-box containing the set of strings defined in the named list is offered to the user.
user	One of the principal-names in the user registry maintained by the AccuRev server.	None. In the edit form, a list-box containing all principal-names is offered to the user.
timestamp	An AccuRev timestamp.	Granularity (year, month, day, hour, minute, or second). In the edit form, the user fills in fields that indicate a time, to the granularity you specify here.
attachment	A set of attachment definitions.	Height, Width: the height (number of lines) and width (approximate number of characters) of the edit-form field that lists the attachment definitions.
internal	A positive integer.	None. You cannot create a field with this data type; it is used only by the built-in fields issueNum and transNum .

Defining Multiple-Choice Lists

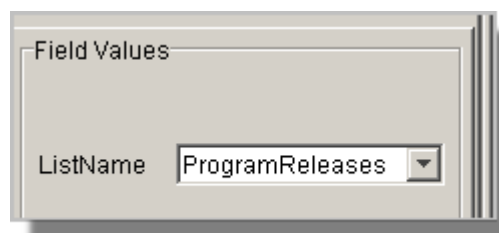
The **choose** and **list** data types are similar:

- For a **choose** field, the set of possible values — an ordered list of strings — is part of the individual field definition. You enter the possible-values list in the Field Values box for that field.



The 'Field Values' dialog box for a 'choose' field. It features a list box on the left containing the values 'high', 'medium', and 'low', with 'high' selected. To the right of the list box are four buttons: 'Add', 'Remove', 'Up', and 'Down'. Above the list box is an empty text input field.

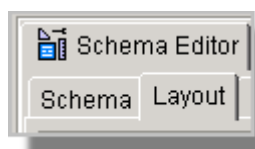
- For a **list** field, the set of possible values is also an ordered list, but it is *not* part of the individual field definition. In the Field Values box, you specify the name of one of the lists created on the Lists subtab. Any number of **list** fields in a Dispatch database can use the same named list:



The 'Field Values' dialog box for a 'list' field. It shows a 'ListName' label followed by a dropdown menu currently displaying 'ProgramReleases'.

The mechanics of defining the ordered list is similar in the “local” case (for an individual **choose** field) and in the “global” case (on the Lists subtab, for use by any number of **list** fields). On the Lists subtab, you must supply a ListName for the list; for an individual **choose** field, the possible-values list doesn’t need or have a name.

Designing an Edit Form



You design the edit form for a Dispatch database on the Layout subtab. An edit form consists of field-labels and corresponding edit-widgets, arranged in rows and columns. The field-labels come from the Labels column of the Schema Fields table (Schema subtab). The edit-widget for a field depends on its data type and, in some cases, on the additional Field Value information.

The screenshot shows an edit form with a grid layout. Red lines indicate the grid structure. Annotations include:

- column**: Two columns are labeled at the top.
- row**: Five rows are labeled on the left.
- field-label**: Labels like "Issue", "State", "Summary", "Problem", "Submitter", "Submitted on", "Program", and "Release" are shown next to their respective edit-widgets.
- edit-widget**: The input fields, dropdowns, and date pickers.
- "Summary" spans two columns**: An arrow points to the "Summary" field, which spans both columns.
- "Problem" spans two columns**: An arrow points to the "Problem" field, which spans both columns.

The Layout subtab provides a drag-and-drop canvas on which you design the edit form's rows and columns. On this canvas, each field-label/edit-widget pair is represented by a yellow box. The following screen shot shows the layout that defines the edit form illustrated above:

The screenshot shows the Schema Editor Layout subtab. Annotations include:

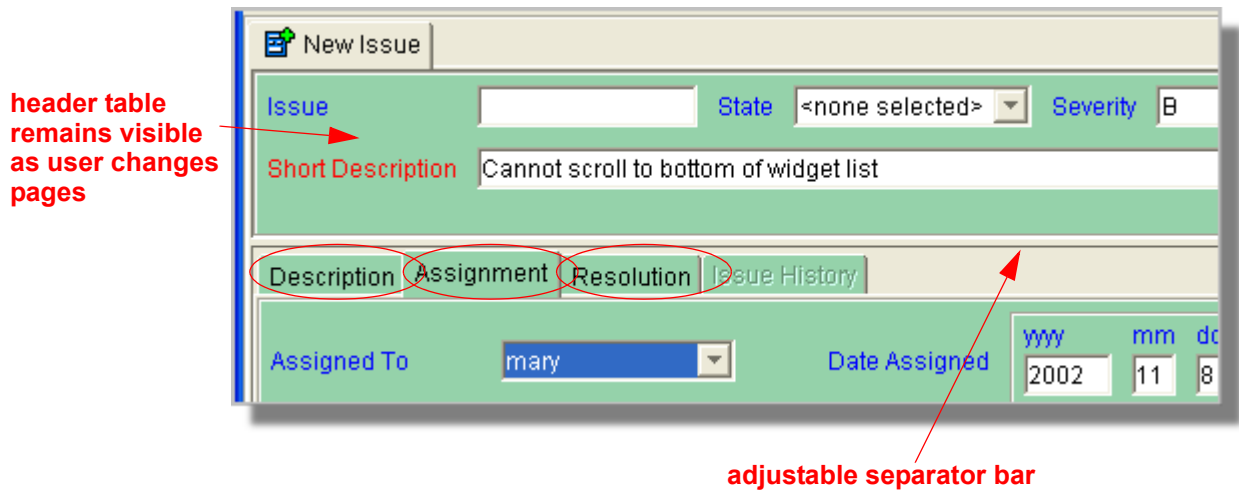
- each "table" defines a separate tabbed page on edit form**: An arrow points to the "Table" header.
- fields that span multiple columns**: Arrows point to the "problem_headline" and "problem_description" fields, which span multiple columns.
- field-name, not field-label**: An arrow points to the "submit_date" field, which is circled.

As the screen-shot annotations indicate, the yellow box contains a field-name, not a field-label. On the edit form, the field-label and the corresponding edit-widget will appear, side by side, at this position. You can also expand a yellow box to make it span two or more columns.

You can organize the fields into multiple "tables", each of which defines a separate tabbed page in the edit form.

The screenshot shows the edit form with two tabbed pages: "Problem" and "Fix". The "Problem" page is active, showing fields like "Assigned to", "Priority", and "Assigned on". The "Fix" page is also visible.

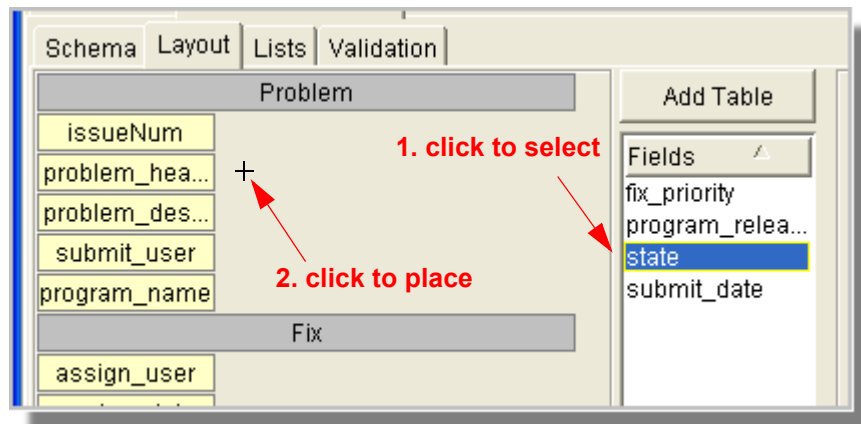
One of the tables can be a header table. Instead of defining a separate tabbed page, a header table defines a set of fields that always appear on the edit form, no matter which tabbed page is currently visible.



Form Layout Operations

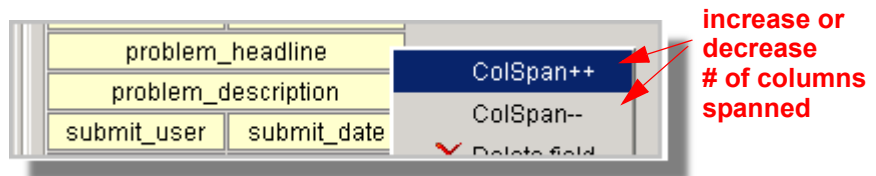
This section describes how to perform the various edit-form design operations on the Layout subtab.

- **Adding a field to the edit form:** The **Fields** list-box offers all fields that do not currently appear in the edit form. Click to select a field in this list-box. Then, click at the empty location on the canvas where you want to place the field.



- **Arranging fields into rows and columns:** Drag-and-drop the yellow boxes to the desired location on the canvas. Drop a box on top of another one to push it rightward or downward.

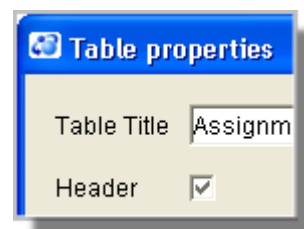
- **Changing column-spanning:** Right-click a yellow box, and select one of the spanning operations from the context menu.



- **Creating a new page in the edit form:** Click the **Add Table** button above the **Fields** list-box. Fill in the Table Title field in the dialog box that appears, then click **OK**. This title appears on the page's tab in the edit form.

A gray box with the specified title appears at the bottom of the layout. Fields that you place below this box will appear on the new page of the edit form.

- **Creating a header table for the edit form:** When creating a new page for the edit form (see above), check the **Header** checkbox if you want it to be the header table. You can also redefine an existing page to be a header table: right-click the gray box and select **Properties**. All fields in the header table will appear at the top of the database's edit form, above every tabbed page.



Remember that an edit form can have at most one header table. Its fields always appear at the top of the edit form, even if it is not the topmost table on the Layout tab.

- **Renaming a page:** Right-click on the gray box, select **Properties**, and change the entry in the Table Title box.
- **Arranging fields into multiple pages:** Drag-and-drop the yellow boxes (fields) and the gray boxes (pages). Fields that you place below a gray box will appear on the corresponding page of the edit form.
- **Removing a field or page:** Right-click the yellow box (field) or gray box (page), and select **Delete** from the context menu. Deleting a page doesn't delete the fields currently on the page — it just deletes the gray box, effectively merging the fields into the page above.

Note: you cannot delete the first gray box; an edit form must have at least one page!

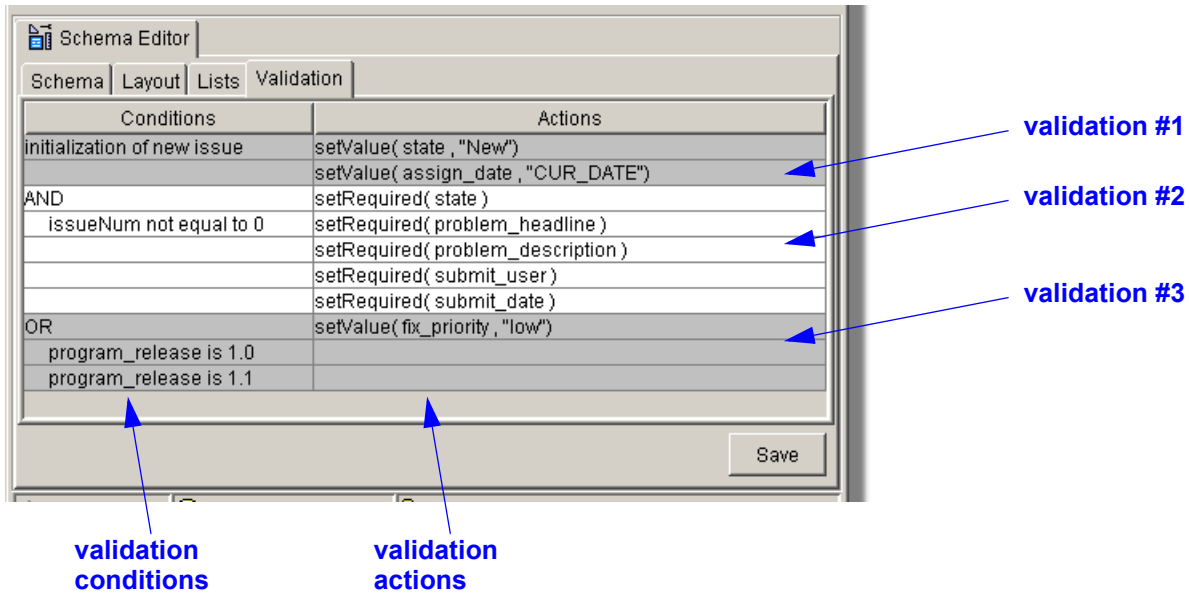
- **Setting the edit form's colors:** Click one of the colored buttons at the right side of the Layout subtab. Then select a color from the **Pick a color** window. You can set these colors:
 - **Foreground:** the color of the edit form's field-labels.
 - **Required Foreground:** the field-label color for fields where a value must be specified. Required fields are defined on the Validations subtab. See *Defining Edit Form Validations* on page 28.
 - **Background:** the color of the edit form's background.

Note: "Button Foreground" and "Button Background" are currently unused.

Defining Edit Form Validations

Users of a Dispatch database create and modify issue records through the database's edit form. To increase the efficiency and accuracy of this process, you can create a set of validations to be applied as the user works with the edit form. (Validations are sometimes called "edit checks".)

You create and maintain the set of validations using a point-and-click interface. Dispatch displays the current validations in tabular format; each one takes the form "if a certain condition is true, then perform a particular set of actions".



The following kinds of validations are available:

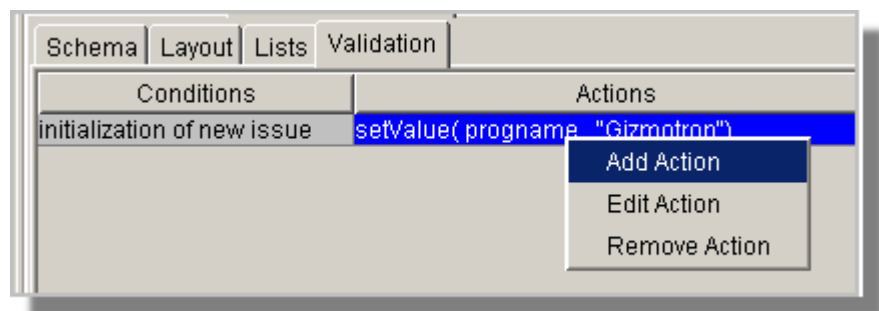
- Initialize a field in a new issue record with a specified value.
- Require a value to be entered in a field.
- Based on a condition, make one or more of these changes:
 - Set a **text** field to a specified value.
 - Set a **user** field to a particular value — for example, to the current user.
 - Set a **timestamp** field to a particular value — for example, to the current time.
 - Make one or more fields into required fields.
 - Change the possible-values list for a **choose** field.
 - Set the entire issue record, a particular page (tab) of the issue record, or a particular field to be read-only.

The condition you specify is essentially the same as a Dispatch query; if the condition is true for (“selects”) the current record, the specified changes are applied to the edit form.

Initializing Field Values in a New Issue Record

The first entry in the table of current validations is special. Its condition is always “initialization of new issue”, so that its actions are performed exactly once: between the time the user invokes the **New Issue** command and the time the new edit form appears.

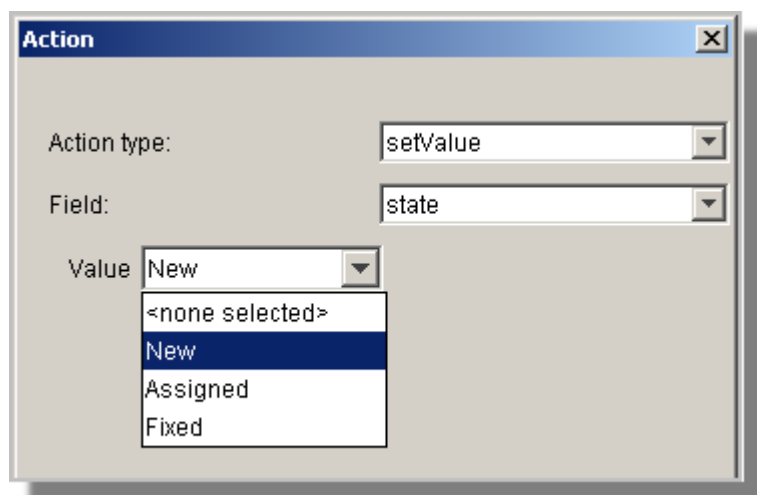
To define an action, right-click in the **Actions** column, then select **Add Action** from the context menu.



An **Action** window appears, in which you define the action. For initialization of a new issue record, the only appropriate action type is **setValue**.

Note: some releases of Dispatch allow you to include **setRequired** and **setChoices** actions in a record initialization. Such actions will be ignored when a new record is initialized.

After setting the action type to **setValue**, select a field to be initialized and specify the initial value. The **Value** edit-widget adjusts to the selected field. In this example, **state** is a “choose” field, so the edit-widget let’s you select one of its predefined choices to be the initial value.



Conditional Validations

The setting of initial field values is an unconditional validation: it happens every time a **New Issue** command is invoked. All other validations are conditional: a certain set of actions are performed if, and only if, a certain condition is met.

The unconditional setting of initial field values occurs just once; but the conditional validations are performed repeatedly: when an edit form first appears *and* each time the user changes any field value. Each repeat involves:

- Clearing the “required” status of all fields.
- Performing all validations (except for that first one: “initialization of new issue”). If a validation’s condition is true, the corresponding actions are invoked.

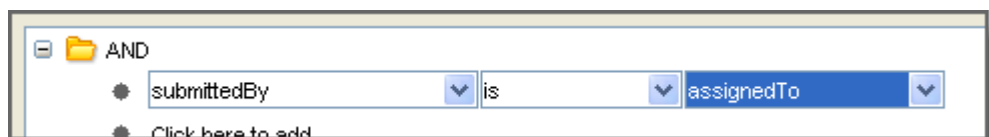
Specifying the Condition

Specifying the condition — the “if clause” — of a validation is very much like specifying a Dispatch database query. (Queries are described in section *Using Database Queries* on page 7.) But there are some differences: in a validation condition, you can:

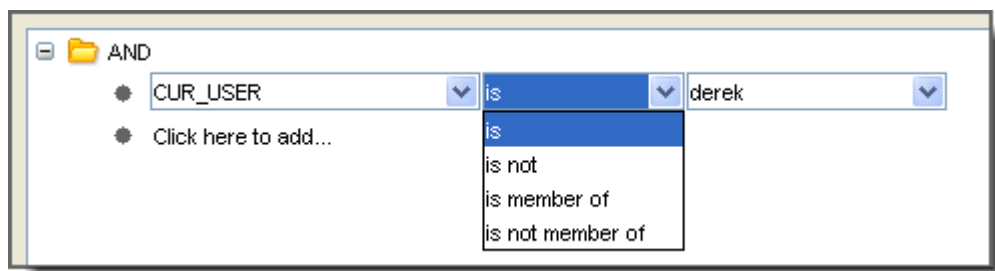
- Compare the current (in memory) value of a field with its “old” (on disk) value — that is, the value currently displayed for the field vs. the value stored in the Dispatch database by the most recent **Save**.



- Compare the (current or old) value of a field with the (current or old) value of another field in the same record.

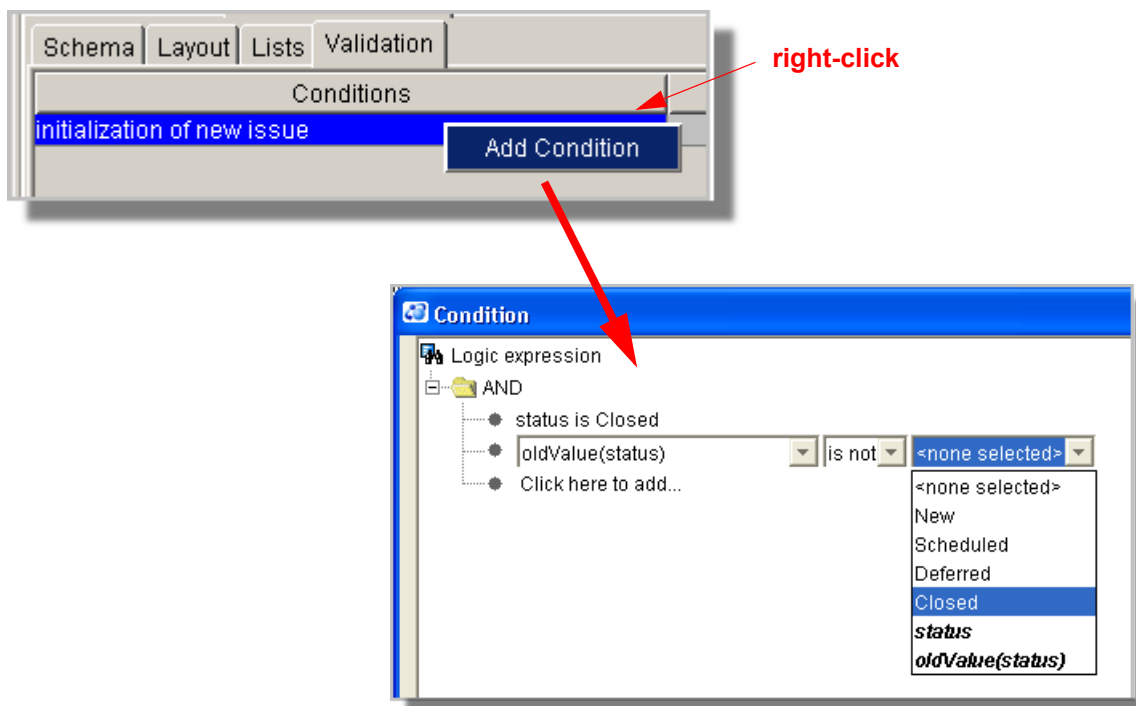


- Test the AccuRev user identity or group membership of the person using the edit form (CUR_USER).



Query conditions cannot make such field-to-field comparisons; they can only compare field values to literal values.

To create a new conditional validation, right-click within any existing condition, and select **Add Condition** from the context menu. This example shows how to define a condition that tests whether the user has changed the value of the **fix_priority** field:



The following sections describe the “then clause” of a conditional validation — the actions to be invoked if the condition is true:

- *Setting a Field Value* (**setValue**)
- *Revising the Choices for a “choose” Field* (**setChoices**)
- *Requiring a Value to be Entered in a Field* (**setRequired**)
- *Setting Permissions on All or Part of the Issue Record* (**setIssuePermission**, **setTabPermission**, **setFieldPermission**)

Each validation can invoke any number of actions, of any type.

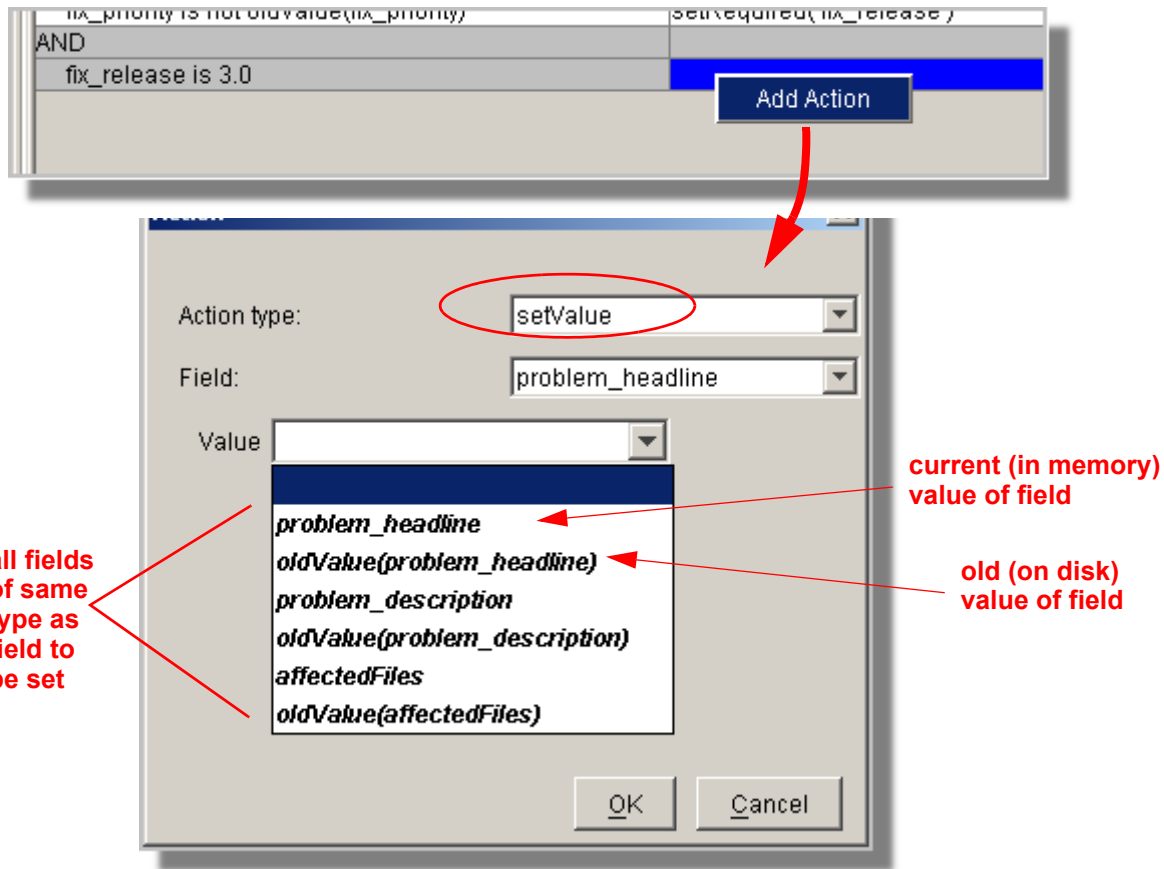
Setting a Field Value

In addition to initializing field values (see *Initializing Field Values in a New Issue Record* on page 29), you can set the values of fields while the user is working with the record, based on certain conditions. For example, if the user enters **ColorStar** in the **program_name** field, a validation could automatically set the **fix_priority** field to **high**. (Management has mandated rapid improvement in the robustness of the ColorStar application.)

To define a **setValue** action for a validation condition:

1. Right-click in the **Action** column next to the condition, and select **Add Action** from the context menu.
2. In the **Action** window that appears, select **setValue** as the action type, and select the field to be set.

3. The **Value** edit-widget adjusts to the selected field, just as it does when you're initializing a field in a new issue record. For a "text" field, you can enter a text string as the value; alternatively, you can open the list-box to specify either the current value or the old value of any of the database's "text" fields (see example). Setting the value for a "timestamp" or "timespan" field works similarly.



For fields of type "choose", "list", or "user", you can use the list-box to specify any of the field's predefined values.

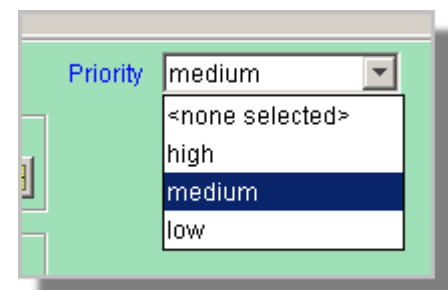
Revising the Choices for a "choose" Field

The definition of each field of type "choose" includes an ordered list of strings. The user fills in this field by choosing one of the strings from a list-box.

In a validation, a **setChoices** action can change the "choice list" that the user will see when he opens the list-box: The changes to the choice list can include:

- Removing one or more of the existing choices
- Changing the order of the choices

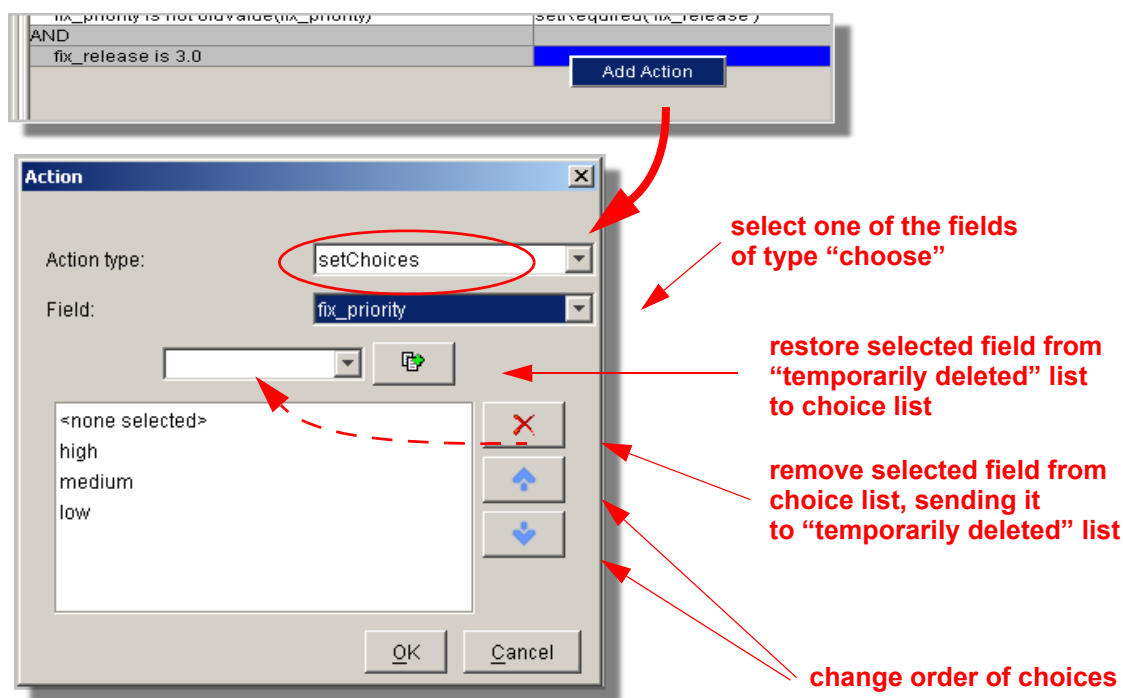
You cannot add new strings to the choice list with a **setChoices** action. The changes made by a **setChoices** action are temporary: they last only until the next **setChoices** action on the same field,



or until the user closes the edit form. When an edit form is opened on another issue record (or subsequently on this issue record), the user will see the field's original choice list, as defined on the **Schema** tab.

To define a **setChoices** action for a validation condition:

1. Right-click in the **Action** column next to the condition, and select **Add Action** from the context menu.
2. In the **Action** window that appears, select **setChoices** as the action type, and use the **Field** list-box to select one of the database's fields of type "choose".
3. Use the controls at the bottom of the **Action** window to define the temporary change to the choice list:



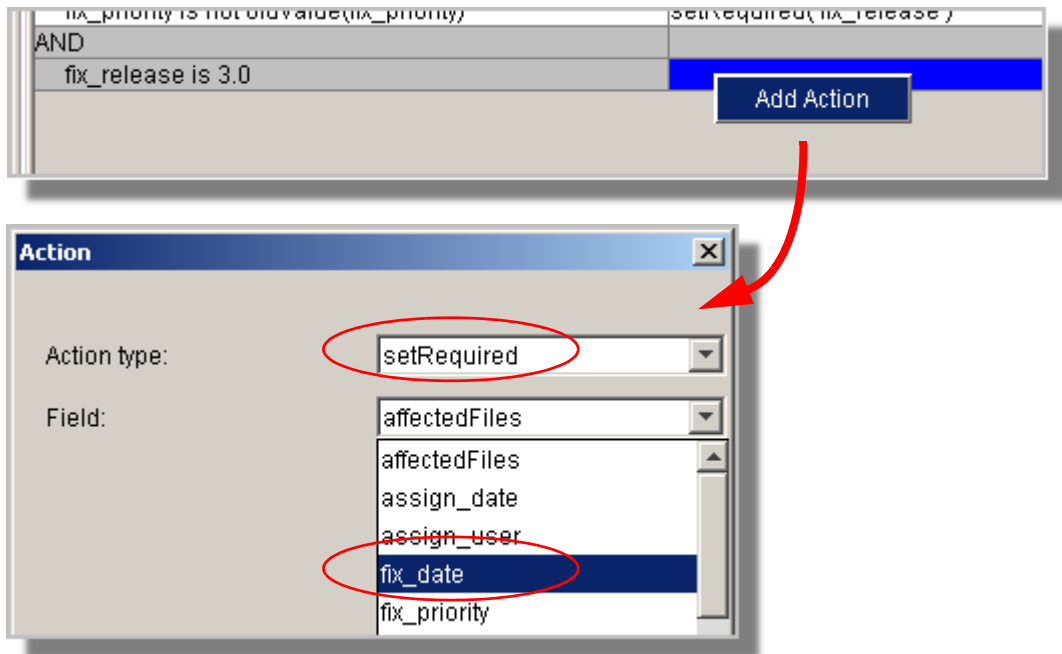
Requiring a Value to be Entered in a Field

Like many other programs that process fill-in-the-blanks forms, Dispatch can treat certain fields as required fields — the user must specify a value in each such field before Dispatch will create or update an issue record. (For a "choose", "list", or "user" field, the value must not be the **<none selected>** placeholder.)

Dispatch affords you great flexibility in controlling required fields. A field's "required" status is not part of the basic database schema. Instead, it is controlled by the conditional-validation facility. Thus, Dispatch repeatedly redecides whether a field is required: when the edit form first appears *and* each time the user changes any field value on the edit form. Whenever the user clicks the **Save** button, Dispatch uses the current set of required fields to allow or disallow the "save" operation.

To define a **setRequired** action for a validation condition:

1. Right-click in the **Action** column next to the condition, and select **Add Action** from the context menu.
2. In the **Action** window that appears, select **setRequired** as the action type, and select the field to be required.



Unconditionally Required Fields

If you want certain fields to be required in all circumstances, use a condition that's always true. For example:

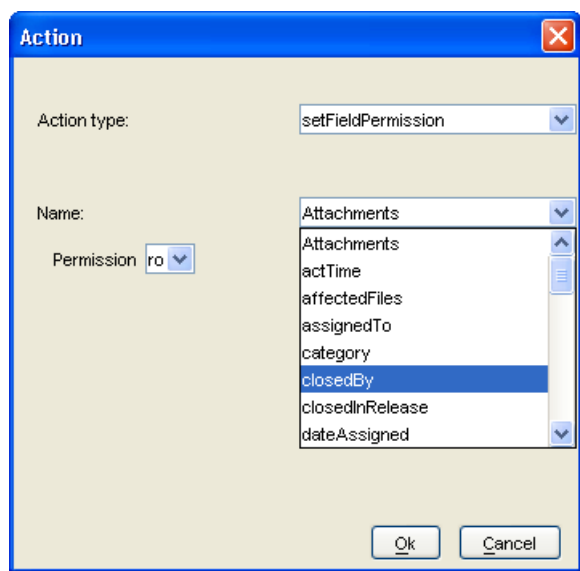
```
issueNum is not 0
```

Setting Permissions on All or Part of the Issue Record

The **setIssuePermission**, **setTabPermission**, and **setFieldPermission** actions are essentially similar: they restrict the editability of some component of the issue record:

- **setIssuePermission** makes the entire issue record read-only.
- **setTabPermission** makes a particular page (tab) of the issue record read-only. See *Designing an Edit Form* on page 25.
- **setFieldPermission** makes a particular field of the issue record read-only.

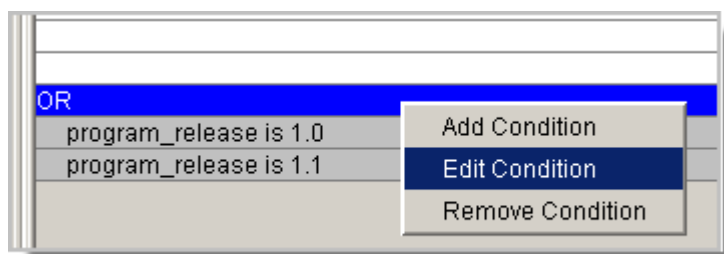
As of Version 3.5, the “read-only” (**ro**) permission is the only one you can set with these commands. This setting affects the user’s current access to the issue record; the “read-only” status is not stored in the repository. For example, user **jjp** might find that he cannot change any fields on the **Assignment** page of any issue record, because of this validation:



This restriction affect **jjp**’s access to issue records; other users’ access remains unaffected.

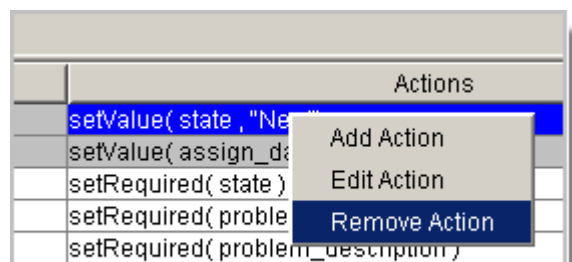
Revising and Removing Validations and Actions

Each conditional validation consists of a condition (left column) and a set of actions (right column). To revise or remove an entire validation, right-click anywhere within the condition, and select the appropriate command, **Edit Condition** or **Remove Condition**, from the context menu.



For the first (unconditional) validation, “initialization of new issue”, you cannot remove or revise the condition. You can only work with the validation’s actions.

To revise or remove an individual action from a validation, right-click on that action and select the appropriate command, **Edit Action** or **Remove Action**, from the context menu.



The Change Packages Tab

As of Version 3.5, AccuRev Enterprise supports an enhanced integration between AccuRev and Dispatch: any Dispatch issue record can act as a change package, keeping track of which versions were created to fix the bug (or implement the new feature) described in that issue record.

In the Schema Editor, the Change Packages subtab provides controls for certain aspects of this facility. For details, see *Viewing Stream Contents/Differences in Terms of Change Packages* on page 43.

Change Packages and Integrations between Configuration Management and Issue Management

Any version-control system must be able to keep track of the changes that developers make to individual files. A full-fledged configuration management system, like AccuRev, should be able to handle questions like these:

“What were all the changes made to source files in order to fix bug #457?”

“Have all the changes made to fix bug #457 been handed off to the QA Group” (That is, have the appropriate versions been promoted to the QA stream?)

Starting in Version 3.5, AccuRev can handle such questions through its change package facility. (Change packages are available in the AccuRev Enterprise product only.) A change package is a collection of element versions; for example:

```
version kestrel_dvt_jjp/13 of element ./src/brass.c  
version kestrel_dvt_jjp/14 of element ./src/brass.h  
version kestrel_dvt_jjp/16 of element ./src/commands.c
```

The basic idea is that this set (or “package”) of versions contains all the changes required to implement a certain development project. But we need to refine this idea. Consider that version 14 of **brass.h** probably contains *more* than just the changes for that development project. For example:

- Versions 1-7 might have been created years ago, when the product was first developed
- Versions 8 and 9 might have been minor tweaks, performed last month
- Versions 10-14 are the only versions with changes for the development project in question

So we need a way to express the idea that only the “recent changes” to **brass.h**, those in versions 10-14, are to be included in the change package. AccuRev accomplishes this by defining each change package entry using two versions: a user-specified head version and an older, automatically-determined basis version. The “recent changes” to be included in the change package were made by starting with the basis version (version 9 in this example) and **Keep**’ing one or more new versions (versions 10, 11, 12, 13, and 14 in this example).

In the AccuRev GUI, the head version of a change package entry is usually identified simply as the “Version”.

Note: the **Patch** command uses the same “recent changes” analysis to determine which changes in the “from” version are to be incorporated into the “to” version.

Where should the change package entry for **brass.h** be recorded? AccuRev already provides a mechanism for keeping track of development projects: the Dispatch issue-management facility. Each project — fixing a bug, creating a new feature, etc. — is tracked by a particular Dispatch issue record. So it makes sense to implement change packages using issue records.

Each issue record includes a **Changes** section that acts as an “accumulator” for versions’ changes. Here’s how the above example of a change package would appear in an issue record’s edit form:

Short Description: Default colors are bogus

Issue: 5 Status: Scheduled State: Open

Severity: D Priority: 1

Basics Assignment Misc Attachments Resolution **Changes** Issue History

Location	Version	Basis Version
/doc/chap02.doc	5/15	4/2
/src/brass.c	5/20	15/1
/tools/cmdshell/end.sh	5/20	5/16

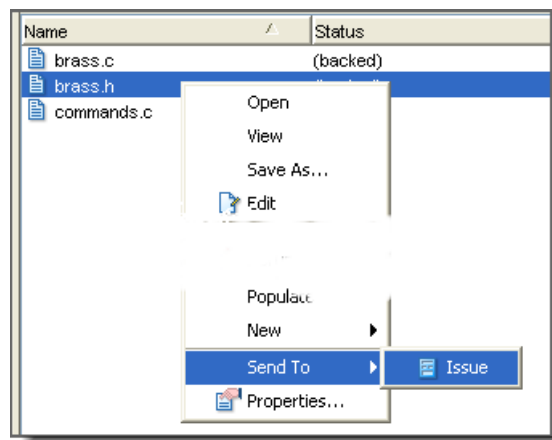
This change package has entries for three elements:

- **chap02.doc**: This change began when the user updated his workspace, bringing in version 4/2 of the element (which had originally been created in another workspace, then was promoted to the backing stream). Then, the user created one or more versions in workspace #5, up to and including version 5/15.
- **brass.c**: Similarly, this change began when the user updated his workspace, bringing in version 15/1, originally created in another workspace. The user created one or more versions in his workspace, ending with version 5/20.
- **end.sh**: The basis version, 5/16 was created in the user's own workspace. This indicates that the user promoted 5/16 to the backing stream. AccuRev assumes that this change was for another task, not the one covered by this issue record. Then, the user turned his attention to the current task, creating versions 5/17 through 5/20.

Each change package can include at most one entry for a given element. This rules helps to ensure that the changes in a given change package are consistent with each other. See *Updating of Change Package Entries* on page 43.

Creating Change Package Entries

You can add entries to a change package manually: right-click a version in the File Browser, Version Browser, or History Browser, and then select the **Send to Issue** command from the context menu. The selected version becomes the head version of the change package entry; AccuRev automatically determines the corresponding basis version. As the examples above suggest, AccuRev uses an algorithm that determines the set of “recent changes” to the element, made in a single workspace.



In the Version Browser, a variant command, **Send to Issue (specifying basis)**, enables you to pick the basis version, rather than allowing AccuRev to determine it automatically.

You can also invoke the **Send to Issue** command on the Changes tab of an issue record. This copies an existing change package entry to a different change package (issue record).

AccuRev can record change package entries automatically, whenever the **Promote** command is invoked in a workspace. For example, suppose issue record #3 represents a particular bug (and its fix). Whenever a developer promotes one or more versions whose changes address that bug, he specifies issue #3 at a prompt. AccuRev automatically creates a change package entry in issue #3 for each promoted version.

Automatic recording of change package entries is enabled through an integration between configuration management and issue management. This change-package-level integration is separate from the transaction-level integration that has long been an AccuRev feature. For more on both these integrations, see [Integrations Between Configuration Management and Issue Management](#) on page 45.

Structure of a Change Package

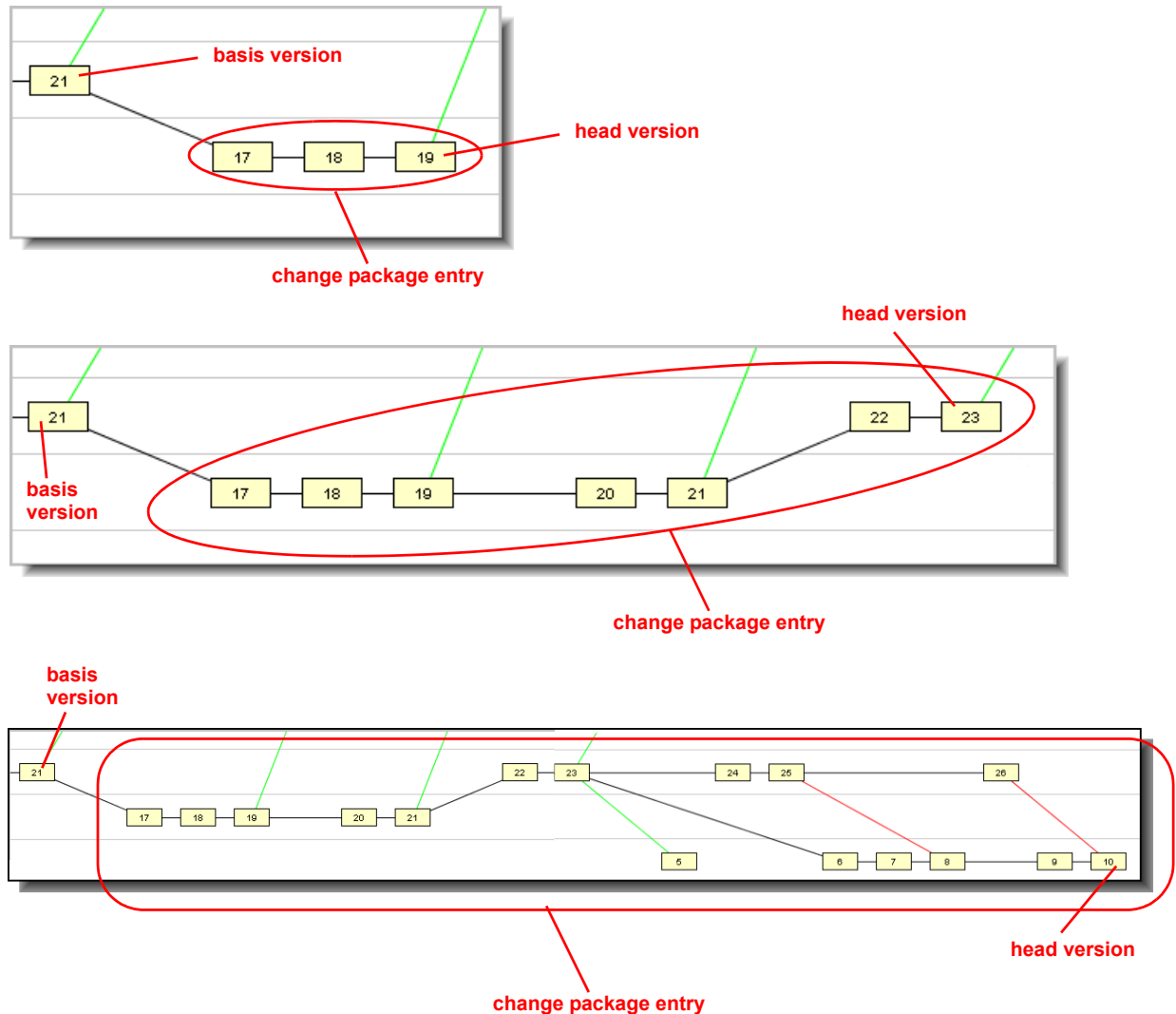
All change package entries are recorded in terms of real versions (those created in users' workspaces), even though there may be corresponding virtual versions (created by promoting the real versions from workspaces to dynamic streams). In all the examples shown above, each change package entry is a series of consecutive real versions created in the same workspace — that is, each change package entry records a particular patch to the element.

But the change package facility can also track *ongoing* changes to elements — changes made at different times, and in different workspaces. To support this capability, AccuRev defines a change package entry in a more general way than a patch:

A change package entry for an element consists of all the real versions in the element's version graph between a specified basis version and a specified head version. Between-ness is determined both by direct predecessor-successor connections (created, for example, by **Keep**)

and by merge connections (created by **Merge**). Patch connections are *not* considered in this determination; the basis version itself is not part of the change package entry.

The following Version Browser excerpts show the range of complexity that a change package entry can have. In fact, these excerpts show how the same change package entry can change over time, becoming more complex. See [Updating of Change Package Entries](#) on page 42.

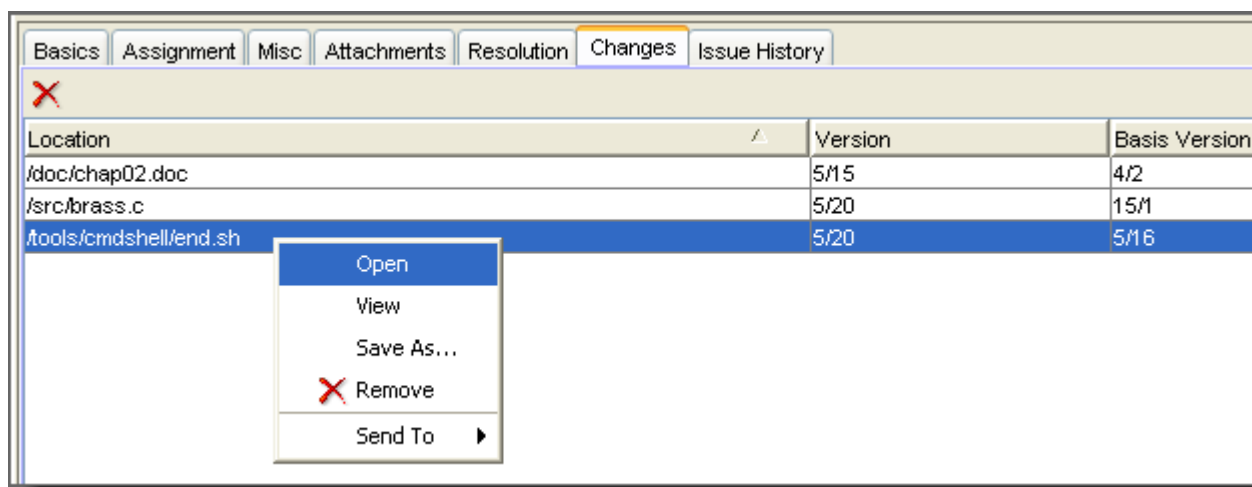


These illustrations suggest the following definition for a change package entry, which is equivalent to the definition above:

A change package entry for an element consists of the element's entire version graph up to the specified head version, *minus* the entire version graph up to the specified basis version. For these purposes, the version graph includes direct predecessor-successor connections and merge connections, but not patch connections.

Operations on Change Package Entries

You can perform several operations on a change package entry, using its context menu. Most of these operations concentrate on the head version, which contains all the changes in the change package entry. (The head version appears in the “Version” column.)



The **Open**, **View**, and **Save As** commands have the same meaning as in the File Browser:

Open

Run the appropriate command on the head version of the change package entry, according to its file type.

View

Open a text editor on a temporary copy of the head version (text files only).

Save As

Copy the head version to another filename.

The **Remove** and **Send To > Issue** commands enable you to maintain and fine-tune the contents of change packages. You can have the same entry(s) appear in two or more change packages: select the entry(s), invoke the **Send To > Issue** command, and specify another issue record. Dispatch offers the set of issue records selected by the default query of the issues database, but you can type in the number of any issue record.

If you want to remove one or more entries from a particular change package, just select them and invoke the **Remove** command. This doesn't affect the entries' membership in other change packages, or the status of their versions in the depot's stream hierarchy.

Updating of Change Package Entries

Change packages are designed to track ongoing changes to elements, not just a single set of changes. This means there will be times when you want to add a change package entry for a particular element, but an entry for that element already exists in the change package. In such

situations, AccuRev attempts to combine the new entry with the existing one, producing an updated entry that includes all the changes. (Recall that there can be at most one entry for a given element in a given change package.)

A Little Bit of Notation

To help explain how AccuRev performs “change package arithmetic” to combine and update entries, we’ll use a simple notation. Suppose a change package entry contains the set of an element’s versions defined by these specifications:

the head version is **H**

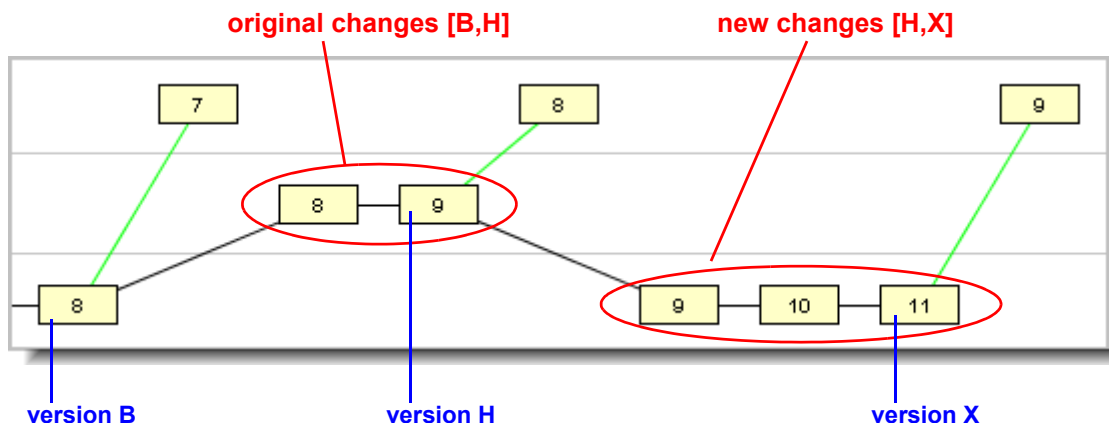
the basis version is **B**

We’ll use the ordered pair **[B,H]** to indicate this change package entry.

Combining Two Change Package Entries

Now, suppose a new change is to be combined with the existing change package entry **[B,H]**. There are several cases, each handled differently by AccuRev:

- **Case 1: [B,H] + [H,X]** — This simple case typically arises when you think you’re done with a task and record your work as change package entry **[B,H]** — but it turns out that more work on the same element is required. So you (or a colleague) start where you left off, with version **H**, and make changes up to version **X**. Then, you want to incorporate the new set of changes **[H,X]** into the same change package.

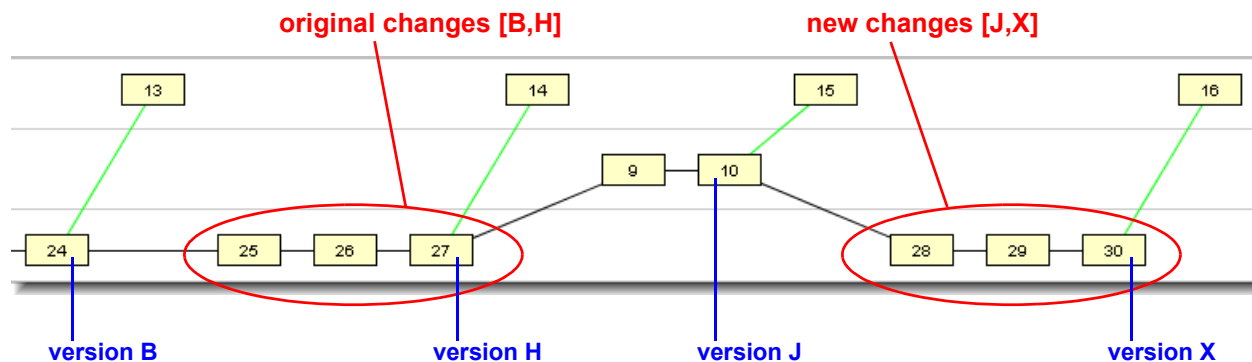


In this case, it’s clear that the two series of changes can be viewed as a single, uninterrupted series — starting at version **B** and ending with version **X**. That is:

$$[B,H] + [H,X] = [B,X]$$

Accordingly, AccuRev updates the change package entry automatically — keeping **B** as the “Basis Version” and changing the “Version” from **H** to **X**.

- **Case 2: [B,H] + [J,X]** (where H is an ancestor of J: “change package gap”) — This case typically arises when you do work on a task at two different times, and someone else has worked on the same element in between.

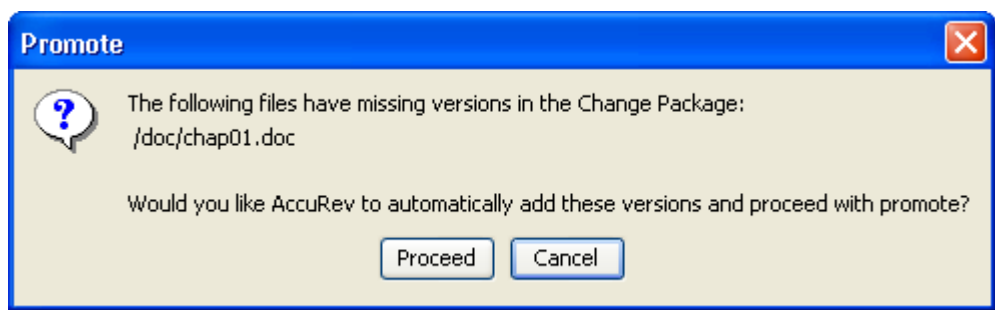


In this example, a colleague updated her workspace to bring in your original changes, created versions 9 and 10 in her workspace, and promoted her changes. You then updated your workspace to bring in her changes, and made a new set of changes.

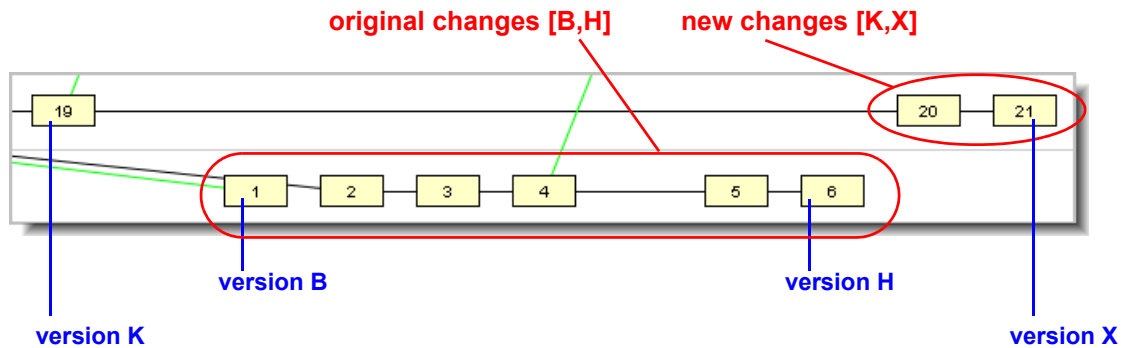
When AccuRev tries to combine the change [B,H] and the change [J,X] into a single change package entry, it detects that version H and version J are not the same, but that H is a direct ancestor of J. Thus, there is a simple “gap” in the potential combined change package entry (in this example, consisting of your colleague’s versions 9 and 10).

Probably, your colleague was not working on the same task when she made her changes. (If she had been, she would have added her changes to the same change package, as in Case 1.) On the other hand, it’s probably OK to include the entire, uninterrupted series of versions [B,X] in your change set — this includes both your original changes and your new changes (and, harmlessly, your colleague’s changes, too).

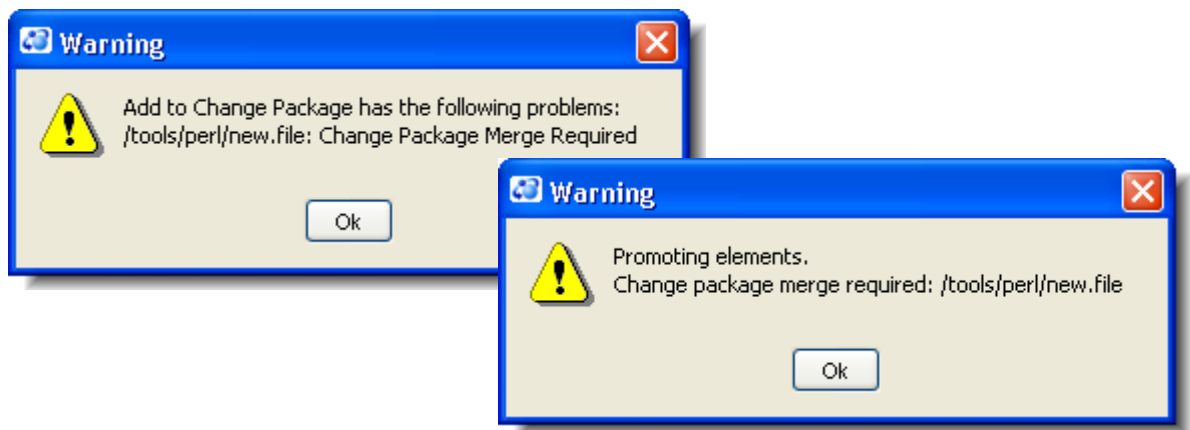
Accordingly, AccuRev prompts you to approve this “spanning the gap” between the two change set entries, in order to create a single, combined entry:



- **Case 3: [B,H] + [K,X]** (where H is *not* an ancestor of K: “change package merge required”) — This case typically arises when developers in workspaces that do not share the same backing stream try to use the same change package. There is no simple “gap” between the existing change package entry and the new one — which means there is no way to combine them into a single change package entry, according to definitions in *Structure of a Change Package* on page 41.



AccuRev signals this situation with a “change package merge required” message, and cancels the current operation.



You can remedy this situation by performing a merge at the element level. (There is no merge operation defined at the change package level.)

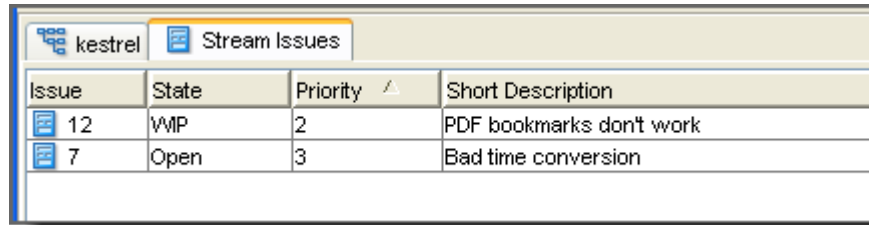
??

Viewing Stream Contents/Differences in Terms of Change Packages

You have always been able to view the contents of a stream or workspace in terms of individual elements and versions (File Browser details pane). Likewise, you’ve been able to compare two workspaces/streams in terms of individual elements and versions (StreamBrowser **Diff** command).

With the advent of change packages, you can now view the contents of a single workspace/stream — or compare two workspaces/streams — in terms of change packages. The commands, available in the StreamBrowser, are **Show Issues** and **Show Difference > By Issues**. For both these commands, the result is a set of issue records. AccuRev displays the results similarly to the way it displays the results of a Dispatch query: a table in which each row is one issue record and each column is one field:

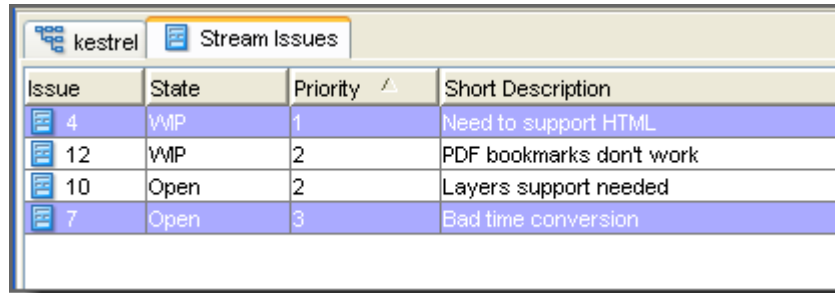
result of
Show Issues



The screenshot shows a window titled 'kestrel' with a tab labeled 'Stream Issues'. It contains a table with the following data:

Issue	State	Priority	Short Description
12	WMP	2	PDF bookmarks don't work
7	Open	3	Bad time conversion

result of
**Show Difference
By Issues**

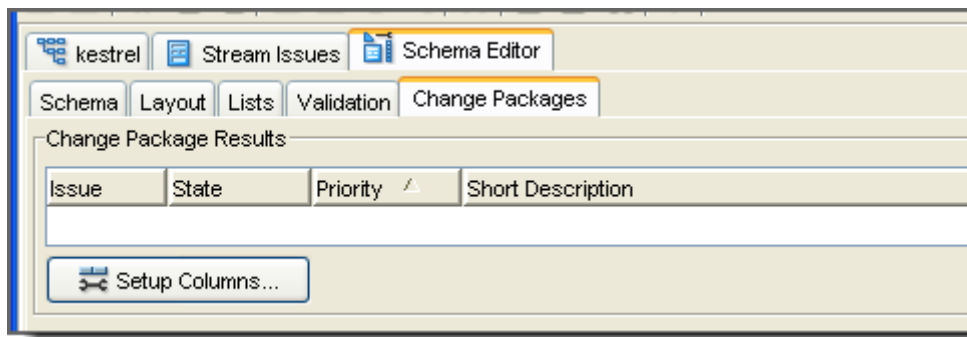


The screenshot shows a window titled 'kestrel' with a tab labeled 'Stream Issues'. It contains a table with the following data:

Issue	State	Priority	Short Description
4	WMP	1	Need to support HTML
12	WMP	2	PDF bookmarks don't work
10	Open	2	Layers support needed
7	Open	3	Bad time conversion

Formatting a Change Package Table

To specify the format of the tables produced by **Show Issues** and **Show Differences by Issues** commands, go to the Schema Editor's Change Packages subtab. In the top section, "Change Package Results", you can determine which fields appear as columns, the column widths, the order of the columns (fields), and the sort order of the rows (records).



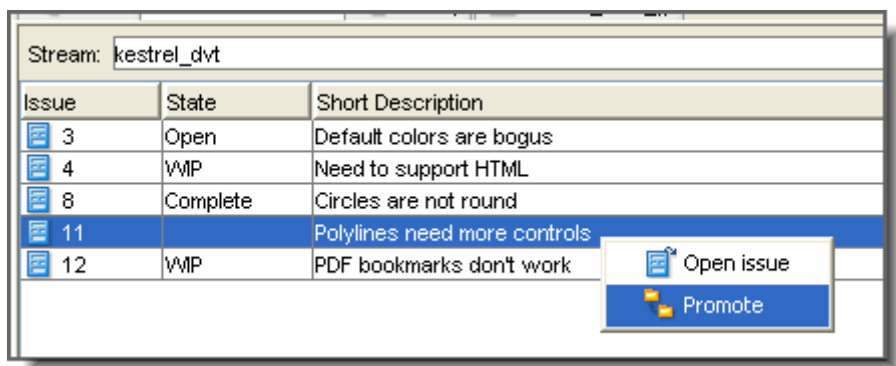
For more on these commands, see the *AccuRev User's Guide (GUI)*: [Viewing a Stream's Change Packages \(Issues\)](#) on page 79, and [Comparing Streams Using Change Packages](#) on page 84.

Promote by Change Package

You can invoke the **Promote** command on one or more of the issue records in a **Show Issues** table. This promotes all the versions in the issue record's change package to the parent stream.

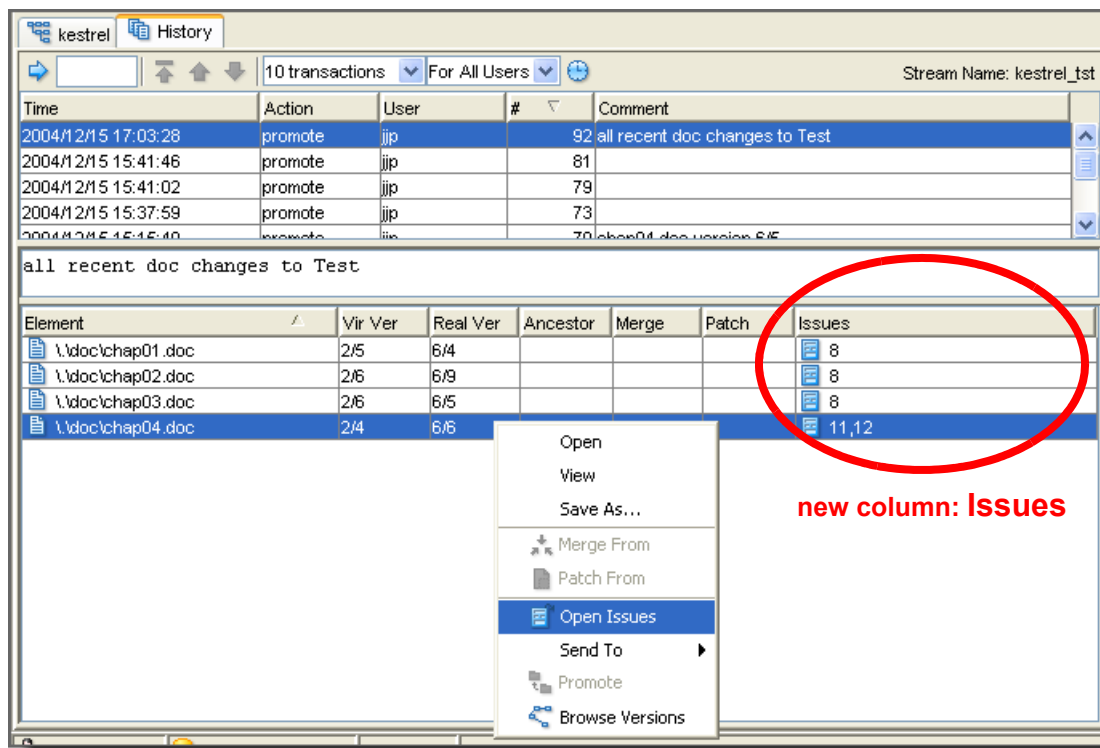
Promotion makes sense only for versions that are currently active in the stream. Accordingly, this command is restricted to issue records that are listed when **Show Active** is checked and **Show Incomplete** is not checked. A version in a change package won't be promoted if AccuRev determines that the changes in that version have already been propagated to the parent stream.

The **Promote** command is *not* enabled for issue records displayed by **Show Difference By Issues**.

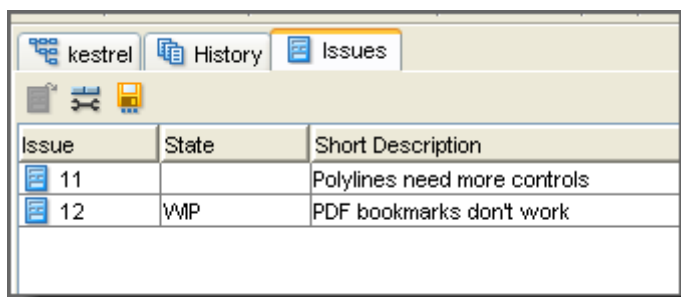


Viewing a Transaction in Terms of Change Packages

The Versions pane of the History Browser shows information on all the versions involved in a selected transaction. This pane now includes an Issues column, which indicates the change package(s) to which each version belongs.



The **Open Issues** command, on the context menu of a version, enables you to view the entire change package(s) — that is, the issue record(s). If you select a version that belongs to a single change package, an edit form opens on that one issue record. If you select a version that belongs to multiple change packages, an Issues tab opens, listing all the issue records.



Integrations Between Configuration Management and Issue Management

This section describes two similar, but separate facilities that integrate AccuRev's configuration management functionality with its issue management (Dispatch) functionality. One of these, introduced in Version 3.5, uses change packages as the point of integration. The other facility, available since Version 3.1, uses a particular issue-record field, **affectedFiles**, as the point of integration.

Both the integrations are activated when a **Promote** command is executed on versions in a workspace. (Promoting versions from a dynamic stream does not activate either integration.) And both integrations record information about the **Promote** transaction in a user-specified Dispatch issue record.

Change-Package-Level Integration

The integration between configuration management and issue management at the change package level was introduced in AccuRev Version 3.5, the first version that featured change packages. When a **promote** command is executed a user's workspace (but not in a higher-level dynamic stream), the integration records all the promoted versions in the change package section of a user-specified Dispatch issue record.

Enabling the Integration

The change-package-level integration is enabled on a depot-by-depot basis. Open the Dispatch Schema Editor for a particular depot's issue database, and go to the Change Packages subtab. Filling in the lower section, "Change Package Promote Triggers", enables the integration for that particular depot.

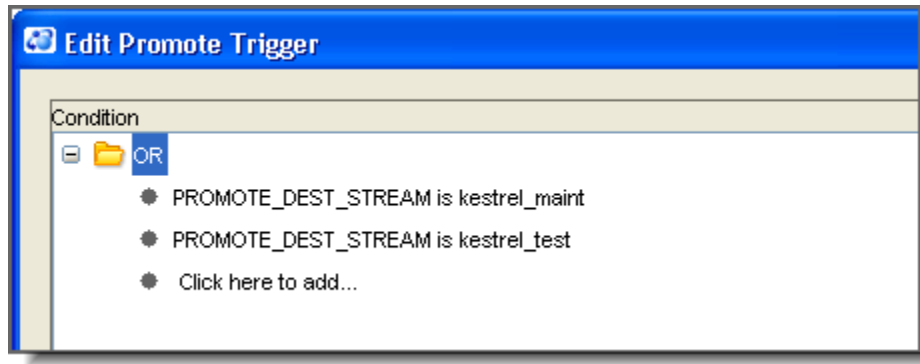
The Change Package Promote Triggers section is structured as a set of condition/query pairs. One of the queries will be selected for execution at **promote**-time. If the first condition is satisfied, the first query will be executed; otherwise the second condition will be evaluated, and if it's satisfied, the second query will be executed; and so on.

Conditions	Queries
AND PROMOTE_DEST_STREAM is in hierarchy Problems_Publisher	AND productType is Publisher type is enhancement
AND PROMOTE_DEST_STREAM is in hierarchy Problems	AND issueNum greater than or equal to 1 type is defect

Issue	Assigned To	Date Assigned	State	Short Description
-------	-------------	---------------	-------	-------------------

Setup Columns...

Each clause of a condition performs a test on the **promote** destination stream. For example, this condition is satisfied if the user is promoting to either of the streams **kestrel_maint** or **kestrel_test**:



The query corresponding to each condition can be any Dispatch query, which selects a set of issue records. See [Using Database Queries](#) on page 7.

The Change Package Promote Triggers section also specifies the format of each query's results table — the fields to appear as columns, the column widths, the order of the columns (fields), and the sort order of the rows (records).

Triggering the Integration

Once the integration is enabled for a depot, it is triggered whenever a user performs a **promote** command in a workspace associated with that depot.

If the **promote** command is invoked through the AccuRev GUI:

1. One of the Dispatch queries specified in the Change Package Promote Triggers section is executed, selecting a certain set of issue records.
2. Those records are displayed to the user in the results table format specified in the Change Package Promote Triggers section.
3. The user selects one of the issue records.
4. The command completes its work.
5. The versions involved in the command are recorded in the change package section (Changes page) of the selected issue record.

If the **promote** command is invoked through the AccuRev CLI:

1. Just as with the GUI, one of the Dispatch queries specified in the Change Package Promote Triggers section is executed, selecting a certain set of issue records.
2. The user is prompted to specify an issue record:

```
Please enter issue number ?
```

Users can bypass this prompt by specifying an issue number with the **-I** option:

```
> accurev promote -I 4761 chap01.doc
Validating elements.
Promoting elements.
Promoted element \.\doc\chap01.doc
```

When updating an existing change package entry, this might cause a “change package gap” situation, which cancels the **promote** command:

```
Promoting elements.  
Change package gap: /doc/chap01.doc
```

In this case, add the **-g** option, in order to “span the gap”:

```
> accurev promote -I 4761 -g chap01.doc  
Validating elements.  
Promoting elements.  
Promoted element \.\doc\chap01.doc
```

See *Updating of Change Package Entries* on page 43 above.

3. Assuming the user-specified issue record is one of those selected by the query, the command completes its work, and the promoted versions are recorded in the change package section of the selected issue record.

What happens if the user specifies no issue record or a non-existent issue record — for example, 99999? In both the GUI and CLI cases:

- If the user specifies no issue record by making a blank entry, the command completes but no information is sent to the issues database. (For this purpose, entering issue record #0 counts as “no issue record”).
- If the user specifies a non-existent issue record, the command is cancelled, and no information is sent to the issues database.

Transaction-Level Integration

The integration between configuration management and issue management at the transaction level was introduced in AccuRev Version 3.1. The integration records the number of each **promote** transaction in a user-specified issue record.

Enabling the Integration

The transaction-level integration is enabled on a depot-by-depot basis, by setting the depot’s **pre-promote-trig** trigger. For example:

```
accurev mktrig -p kestrel pre-promote-trig client_dispatch_promote
```

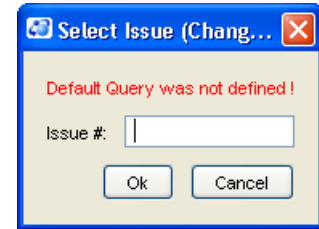
Note that “client_dispatch_promote” is simply a keyword, not the name of a script file. The integration is implemented by two cooperating routines, one built into the AccuRev client software, one built into the AccuRev server software.

Activating the Integration

Once the integration is enabled for a depot, it is activated whenever a user executes the **promote** command in any workspace or dynamic stream.

1. The depot’s default query, as defined on the Queries tab (**Issues > Queries**), is executed and the results are displayed to the user.

2. The user selects one of the issue records. Note that if no default query is defined for the depot, the user is prompted to type an issue record number.
3. The **promote** command completes its work, propagating the versions to the backing stream.
4. The **promote** transaction number is recorded in the **affectedFiles** field of the selected issue record.



Over time, the **affectedFiles** field of a given issue record can accumulate a SPACE-separated list of **Promote** transaction numbers.

Implementation and Customization of the Transaction-Level Integration

When the transaction-level integration is activated, processing takes place on both the AccuRev client machine and the AccuRev server machine:

- The client-side processing — querying the Dispatch issues database and prompting the user to specify an issue record — is structured as a **pre-promote-trig** trigger routine built into the AccuRev client software.
- The server-side processing — updating of the Dispatch issue record — is structured as a **server-post-promote-trig** trigger routine built into the AccuRev server software.

You enable the integration by setting the **pre-promote-trig** trigger with the “client_dispatch_promote” keyword, as described above. You don’t need to explicitly set a **server-post-promote-trig** trigger script. If you do, the script runs instead of — not in addition to — the server-side built-in routine.

In most cases, you’ll want to avoid setting a **server-post-promote-trig** trigger script, just letting the built-in routines do their work. But suppose that after a **Promote**, you want the server machine to perform operations in addition to those defined in the transaction-level integration — for example, updating reference trees and sending email messages. In such cases:

1. Create a script that performs the server-side part of the transaction-level integration, along with the desired additional processing. Start with the sample script **server_dispatch_promote_custom.pl**, which is located in the **examples/dispatch** subdirectory of the AccuRev installation directory.
2. Place the script in the AccuRev **bin** directory.
3. Use a **mktrig** command to make the script the depot’s **server-post-promote-trig** trigger script.

Further customizations of the transaction-level integration are possible. For example, you might want the user to be able to specify several issue records, not just one. Or you might want to link **promote** commands in one depot with the Dispatch issues database in another depot. Or you might want to update an issue record field other than **affectedFiles**. In such cases, you’ll want to dispense with the built-in “client_dispatch_promote” routines altogether:

1. Start with the sample script **client_dispatch_promote_custom.pl** (in the **examples/dispatch** subdirectory), and create your own script for use as a **pre-promote-trig** script to execute on the client.
2. As described above, start with the sample script **server_dispatch_promote_custom.pl** (in the **examples/dispatch** subdirectory), and create your own script for use as a **server-post-promote-trig** script to execute on the server.

If Both Integrations are Enabled

Both the change-package-level and transaction-level integrations can be enabled for a given depot at the same time. In this case, a user performing a **Promote** command in a workspace is prompted to specify an issue record just once, not twice. The prompting for an issue record by the change-package-level integration takes place as usual. That issue record is then updated by both integrations.

Note that even if both integrations are enabled, a **Promote** command performed in a dynamic stream (not a workspace) activates just the transaction-level integration, not the change-package-level integration.