Troubleshooting Specialization

[vertical list of authors]

© Copyright ,.

[cover art/text goes here]

Contents

The DITA troubleshooting specialization

Troubleshooting topics provide descriptions of and solutions or workarounds for problem situations that users might encounter. You create troubleshooting topics by using the DITA troubleshooting specialization, which is derived from the generic topic and includes special elements for troubleshooting information.

Users refer to troubleshooting information to understand what condition or event generated their problem situation and to find out what they need to do to recover from or work around a problem, or to prevent a recurrence of the problem. Typically, troubleshooting information includes a description of the event that generated the problem, the symptoms, the environment, the possible causes, and suggestions for recovery actions. The specialized troubleshooting topic contains details to help identify the particular problem and includes responses for different types of users.

Overview of using the DITA troubleshooting specialization

Suggestions for using the DITA troubleshooting specialization are based on industry-wide research, best-of-breed architecture recommendations, and numerous reviews. You can increase the quality, consistency, and retrievability of troubleshooting information by using the troubleshooting specialization and the accompanying usage and reference information.

What is supported

The DITA troubleshooting specialization is designed to be flexible so that it can be effectively used to address the different characteristics of different problem situations, including:

- Problem situations that have multiple causes or for which the cause is unknown.
- Problem situations that have a limited amount of problem-determination information.
- Problem situations that have multiple parallel resolution procedures that might necessitate separate child topics.
- Reuse of existing information. For example, existing task information can be reused and incorporated into the overall troubleshooting information.

Components of a troubleshooting topic

A superior troubleshooting topic depends on the quality and accuracy of the information that is provided. However, including all or a combination of the following elements with the corresponding meaningful information can help you reach that goal:

Table1. Components of a troubleshooting topic

Type of information	Generated label, if any	Element name
A meaningful title	N/A	<title>
A short description	N/A	<shortdesc>
A symptoms section	Symptoms	<tsSymptoms>
A causes section	Causes	<tsCauses>
An environment section	Environment	<tsEnvironment>
A diagnosing section	Diagnosing the problem	<tsDiagnose>
A resolving section	Resolving the problem	<tsResolve>
User response defined by role (1 of 12)	Administrator response:	<tsAdministratorResponse>
User response defined by role (1 of 12)	Application programmer response:	<tsApplicationProgrammerResponse>
User response defined by role (1 of 12)	Database administrator response:	<tsDatabaseAdministratorResponse>
User response defined by role (1 of 12)	Hardware service provider response:	<tsHardwareServiceProviderResponse>
User response defined by role (1 of 12)	Network administrator response:	<tsNetworkAdministratorResponse>
User response defined by role (1 of 12)	Operator response:	<tsOperatorResponse>
User response defined by role (1 of 12)	Programmer response:	<tsProgrammerResponse>
User response defined by role (1 of 12)	Security administrator response:	<tsSecurityAdministratorResponse>
User response defined by role (1 of 12)	Software service provider response:	<tsSoftwareServiceProviderResponse>

Type of information	Generated label, if any	Element name
User response defined by role (1 of 12)	System administrator response:	<tsSystemAdministratorResponse>
User response defined by role (1 of 12)	System programmer response:	<tsSystemProgrammerResponse>
User response defined by role (1 of 12)	User response:	<tsUserResponse>
A user response section that you define. Not one of the pre-defined user response roles	N/A	<tsResponseRole>
Customized label that identifies the user role	The label that you specify appears	<tsResponseRoleLabel>
A section that describes the customized user actions	N/A	<tsResponseRoleAction>

With the exception of the tsResponseRole and associated elements, you cannot change the labels for these sections. These labels are intended to improve consistency and retrievability across all troubleshooting information.

Samples

You can find sample files of troubleshooting content in your `samples` directory.

Structure of troubleshooting information

The troubleshooting topic architecture assumes a basic sequence. Some parts of the sequence are optional, but the structure of the troubleshooting topic ensures that writers include the elements that they do use in a consistent order.

Basic troubleshooting sequence

The basic sequence of the troubleshooting topic is:

1. Title.
2. Short description or abstract.
3. Symptoms that the user experiences in the <tsSymptoms> element. <tsSymptoms> is required.
4. Causes of the problem in the <tsCauses> element. <tsCauses> is optional.
5. A description of the environmental characteristics that are associated with the problem in the <tsEnvironment> element. <tsEnvironment> is optional.
6. Information about how to diagnose the problem. This information can be in the <tsDiagnose> element or in a task topic that is imbedded in or referenced from the troubleshooting topic. <tsDiagnose> is optional, although you should omit it only when you include information about diagnosing the problem in an imbedded or referenced task topic.
7. Information about how to resolve the problem. This information can be in the <tsResolve> element or in a task topic that is imbedded in or referenced from the troubleshooting topic. <tsResolve> is optional, although you should omit it only when you include information about resolving the problem in an imbedded or referenced task topic.

Maintaining the sequence

Imbedded or referenced task topics must always come after the <tsDiagnose> and <tsResolve> sections, if those are used. Therefore, if your diagnostic information is in a task topic, you should not use the <tsResolve> element because it would break the sequence and confuse the user. Instead, place all your information about resolving the problem in an imbedded or referenced task topic, even if the quantity and type of information would not normally be enough to justify using a topic. In such a situation, the imbedded topic might include only a short description, or only a single step, as in the following example.

```
<tsTroubleshooting xml:lang="en-us">
.
.
.
<tsBody>
.
.
.
</tsBody>
<task conref="diagnosethisone.dita id=diagnose1"></task>
<task id=resolve1>
<title>Resolving the problem</title>
<taskbody>
<steps>
<step><cmd>Reboot your system.</cmd></step>
</steps>
</taskbody>
</task>
</tsTroubleshooting>
```

Note that in this example, the diagnostic task is called by reference, but the resolution task is directly imbedded because it is so simple.

When to create a troubleshooting topic versus a technote

Some problem-solution information is better suited for technotes than for information center troubleshooting topics. When documenting a troubleshooting topic, consider important criteria that can help you decide where best to publish it.

Today, most products deliver two types of problem-solution documents: Troubleshooting topics and technotes. Troubleshooting topics are developed by information developers who use the DITA troubleshooting specialization; these topics can be made available in the “Troubleshooting and support” node of an information center navigation tree. Technotes are generally developed by subject-matter experts (in Support, Development, Test, or Information Development). These topics can be made available in the appropriate product-specific Support sites. In all other respects, these two document types have the same purpose, and they should have the same structure. If a user fails to complete a task, they need access to both sets of content at the same time and they do not care who wrote the content or how or where it was published.

When documenting a problem and its solution, decide whether to use a troubleshooting topic or a technote based on these factors:

- **Longevity**. How soon do you expect the problem to be fixed? Use a troubleshooting topic if the problem is likely to persist at least into the next product release, or if the problem describes a permanent limitation or restriction or a common but major user error. Use a technote if Support has already published a code fix, or if a fix is likely to be published before the next release.
- **Scope**. How many users are likely to be affected by the problem? Use a troubleshooting topic in an information center if the problem is common or if it affects a large number of users. Use a technote if the problem is unique or if it affects a small percentage of users. Also use a technote to document problems that involve untested configurations.
- **Timing**. At what point in the product lifecycle was the problem discovered? In some cases, you need to use a technote because the information center cannot be updated. Be sure to implement a closed-loop technote process between Support and Information Development to ensure that technote content is moved, where appropriate, into troubleshooting topics for the next information center refresh.

Effective troubleshooting titles

Search title: You can use the searchtitle attribute to make search hits more precise. For example, if you are in the "Setting up logical partitions" topic and want to add a topic on "Troubleshooting connectivity", you might want to add in the searchtitle: "Setting up logical partitions: troubleshooting connectivity".

The titles of troubleshooting topics should be consistent with titles of other troubleshooting topics for the same product or solution. A best practice recommendation is to use task-oriented style heading.

If you are writing new troubleshooting topics or moving existing information into the DITA troubleshooting topic, first look at the current inventory of troubleshooting topics and decide which style of title to use:

- A task-oriented heading that begins with a gerund, such as "Resolving" or "Correcting." This heading seems appropriate because each troubleshooting topic not only includes the reference information that describes the problem, but it also includes task information to help users resolve or work around the problem. This title style is the best practice unless existing troubleshooting information uses the second style.
- A noun string that captures the critical descriptive information about the product.

Regardless of the style of title that you use, try to write the title to maximize the odds that users who have this problem will find this topic.

Short descriptions in troubleshooting topics

The <shortdesc> element is highly recommended for troubleshooting topics. The short description must make sense both as the opening text of the topic and as pop-up text over a related link. Users must be able to determine whether this is the document they need, based on the title and the short description.

Length

Use one to two sentences or 25 to 50 words.

What to say

The short description should briefly describe the problem that the topic is about. If possible, include an action-oriented statement that sums up the approach to resolution; that way users will know that this is more than just a topic that describes the problem. Do not simply repeat the title. Try to include information that will help users understand when the information is appropriate or why the information is necessary, and use terms and phrases that customers are likely to use when searching for this topic.

Phrasing

Do not use sentence fragments; use complete sentences. Avoid starting short descriptions with phrases such as "This topic describes . . . ." or "This topic is about . . . ."

Examples of short descriptions

Changing your password using ldapmodify

If you are logged into the Web Administration Tool, and you change the password using the ldapmodify command, the Web Administration Tool changes the server status to stopped.

Opening additional login panels

If you try to open additional login panels from the File options of the browser and the panel fails, start a new instance of the browser, and open another login panel.

Removing installation lock files

If you attempt to install the product, and a “lock not allowed” message is displayed, remove the lock files that are created to prevent multiple installation processes from running simultaneously.

Tips for making troubleshooting topics retrievable

Writing high-quality troubleshooting information is important, but part of this writing process is making sure that users can quickly find the topics for the problems that they experience.

To minimize the time that a user spends finding a topic for the relevant problem, follow these basic tips for making your topic retrievable:

- In titles, short descriptions, and abstracts include words that users are most likely to search on.
- In titles, put the most important words early in the title so that they can be seen in search results lists, and so that search engines place the topic closer to the top of the results lists.
- Avoid using your company name in the title, short description, or abstract because including the company name probably will not help users find your topic and might result in too many results to review.
- Include the relevant product name in the title, short description, abstract, or all of those tags because seeing the product name can help users know whether this topic is relevant to their problem. For example, if your title is something like "Recovering from database failure," how can users know what database the topic is about?
- In titles, short descriptions, or abstracts, use the plural form of nouns because searches for the singular terms find the plural counterparts, but searches for the plural terms sometimes fail to find the corresponding singular forms.

Task information in troubleshooting topics

Troubleshooting topics provide task information to help users resolve or work around problems, and they sometimes provide task information to help users diagnose the problem. You can include task information in troubleshooting topics in a variety of ways, depending on the characteristics of the information.

The ways in which you can include task information in a troubleshooting topic include:

- Using the optional <tsDiagnose> element, the optional <tsResolve> element, or both elements to provide brief task information in paragraph or list format.
- Using a conref to a reusable task topic that exists elsewhere or that can be reused elsewhere.
- Linking to another task topic that exists elsewhere.
- Imbedding a task topic that provides the relevant steps for users to take.

You are not limited to a single diagnosing task and a single resolving task in a troubleshooting topic. Creating topics with multiple parallel task subsections is appropriate for many complex problems.

The various methods for including task information in troubleshooting topics are described in the following sections.

Include task information in paragraph format

This approach is appropriate when the task information is brief and appropriate for a paragraph or paragraph-equivalent format. (Paragraph-equivalent format refers to paragraphs, lists, and notes.) Only paragraph-equivalent elements can be included in the <tsDiagnose> and <tsResolve> elements. In addition, this approach is appropriate when the task information is very unlikely to be reusable in another context.

This approach is not appropriate if the information:

- Is lengthy, belongs in a step-list format, or requires other elements that are available only in task topics. If any of these conditions exist, use one of the other three approaches.
- Is actually or potentially reusable. If the information is or could eventually be reusable, the conref or linking approach is preferable.

Examples of task information in paragraph-equivalent format:

Here are some examples of paragraphs that might be appropriate in a <tsDiagnose> or <tsResolve> element:

Diagnosing the problem (<tsDiagnose> element)

To determine the appropriate resolution procedure to follow, look at the value in the XYZ field of the error message. If the value is zero (0), follow the procedure “Resolving primary disk failure.” If the value is not zero (0), follow the procedure "Resolving secondary disk failure."

Resolving the problem (<tsResolve> element)

Reboot the system.

Resolving the problem (<tsResolve> element)

Perform one of these actions:

- Run the CLEANUP utility, specifying the correct SYSTEM parameter, and restart the program.
- Run the RESTART utility, and then run the COPY utility for the entire program.

The text that you provide in a <tsDiagnose> or <tsResolve> element can be longer than what is shown in these examples, but if it can be more effective in a procedural (step-list) format, use one of the other methods, each of which supports step lists.

Use a reusable object (task topic)

This approach is appropriate when the task information already exists in a DITA task topic or when the information that you write can potentially be reused elsewhere. This approach can be used when the information belongs in a step list and also when a paragraph structure is sufficient to convey a brief imperative for the user to perform, as long as the content is reusable in whatever form it exists and is referred to with a conref.

This approach is not appropriate if the information:

- Is not reusable now or in the foreseeable future. If this is the case, use either the first approach (include task information in paragraph-equivalent format) or the last approach (directly imbed a task topic).
- Does not need to be included (displayed) as part of the troubleshooting topic. If this is the case, the linking approach is preferable.

How you take advantage of this approach depends on whether the information already exists and, if it does exist, whether it has been made available in a common location as a reusable object. (Reusable objects must have an ID, which all existing task topics have. In addition, reusable objects should generally be placed in a common location for reusable information so that other writers can easily take advantage of their existence.)

Determine which of the following sets of characteristics applies to the information that you are working on:

- If the information already exists in a task topic that is not available in a common location with other reusable objects, work with the writer of the other task topic to place the reusable object in a common location so that you and any other writers can access it. Then you can include the conref to the reusable object immediately after the </tsBody> tag.
- If the information already exists in a common location for reusable objects, specify the conref to the reusable object immediately after the </tsBody> tag.
- If the information does not already exist, but you expect that it can be reused, write the task topic as you normally would, and place it in the common location for reusable objects. Then specify the conref to the reusable object immediately after the </tsBody> tag.

Whenever you use the reusable object approach, the respective <tsDiagnose> or <tsResolve> element is omitted from the topic. For example, if the content of the reusable object helps users diagnose the problem, you do not use the <tsDiagnose> element in the topic. Likewise, if the content of the reusable object helps users resolve the problem, you omit the <tsResolve> element from your topic.

If you use a reusable object in place of the <tsDiagnose> element, you must use either a reusable object or imbedded task in place of the <tsResolve> element. For example, you might have this sort of structure:

```
<tsTroubleshooting>
<title></title>
<shortdesc></shortdesc>
<tsBody>
<tsSymptoms></tsSymptoms>
<tsCauses></tsCauses>
<tsEnvironment></tsEnvironment>
</tsBody>
A conref to a reusable object in place of <tsDiagnose>
Either conref to reusable object or an imbedded task topic in place of <tsResolve>
</tsTroubleshooting>
```

When you use a conref to include a reusable object in place of <tsDiagnose> or <tsResolve>, the related links for the troubleshooting topic are generated in the output immediately prior to the first imbedded topic.

Examples of task information in reusable objects:

Information about how to define and use reusable objects is available in the DITA usage information. Here is an example of what the resolution part of a troubleshooting topic might look like if the information existed in a predefined reusable object.

Figure1. Example of how to include a reusable task in a troubleshooting topic

```
<tsTroubleshooting xml:lang="en-us">
.
.
.
<tsBody>
.
.
.
</tsBody>
<task
conref="commonterms.dita#troubleshooting/resettingapendingstatus"></task>
</tsTroubleshooting>
```

Here is the tagging of this reusable task topic:

Figure2. Sample tagging for a reusable task inside a troubleshooting topic

```
<task id="resettingapendingstatus" xml:lang="eu-us">
<title>Resetting a pending status</title>
<shortdesc>When a utility problem occurs, the utility status is set to a
pending
status, indicating that you must take some action before the system can
operate
normally. Resetting a pending status generally involves running one or
more
utilities.</shortdesc>
<taskbody>
<context>To remove various pending statuses:</context>
<steps>
<step><cmd>Run the REORGANIZE SYSTEM utility to remove the REORGSYS
status.</cmd></step>
<step><cmd>If the system is in auxiliary CHECK-pending
status:</cmd></step>
<choices>
<choice>For user data, run the CHECK DATA utility.</choice>
<choice>For index data, run the CHECK INDEX utility.</choice>
</choices>
</step>
<step><cmd>Run the RESET DATA utility to remove the CHECK-pending
status.</cmd></step>
<step><cmd>Restart the system.</cmd></step>
</steps>
</taskbody>
</task>
```

Link to an existing task topic

Like the conref approach to including task information, linking to another DITA task topic that exists elsewhere is a form of reuse, mainly because it avoids the need to write and maintain information in multiple places. This approach is appropriate when the task topic exists somewhere else and it is long or complex enough that you don't want to actually include the information inside this troubleshooting topic.

This approach is not appropriate if the information:

- Is not reusable in other contexts and is not likely to be reusable in the future. If this is the case, use either the first approach (include task information in paragraph format) or the last approach (directly imbed a task topic).
- Needs to be included (displayed) as part of the troubleshooting topic. If this is the case, use the conref approach instead.

To use this approach, handle the linking to the separate task topic just as you would any other links. The strongly preferred method is to use relationship tables (reltables), but some teams use the related links at the bottom of topics instead. Either method is supported, but, for the sake of consistency, use whatever method is standard for your project.

Whenever you use the linking approach, you can decide whether to include or omit the respective <tsDiagnose> or <tsResolve> element from the topic. For example, if the task topic that you link to helps users diagnose the problem, you might choose not to use the <tsDiagnose> element in the topic. Likewise, if the task topic that you link to helps users resolve the problem, you might choose to omit the <tsResolve> element from your topic.

If you decide to link to an existing DITA task topic that provides information to help users diagnose or resolve the situation, use one of the following approaches:

- Include the <tsDiagnose> or <tsResolve> element in your topic, and use an imperative verb to let the user know what sort of diagnosis task to perform; then include the link to that topic. For example, if the resolution task that you link to has a title of “Coordinating unresolved work,” your <tsResolve> element could simply state, “Coordinate any unresolved work so that you can restart the system.” This approach is generally preferable so that users know to look for the link.
- Omit the <tsDiagnose> or <tsResolve> element from your topic, and include the link to the relevant task topic. The risk with this approach is that users might not realize that a link to important information is available.

Examples of linking to another task topic:

The actual implementation of links, through use of reltables or related links, is well explained in the DITA usage information. Here are a few titles of topics that a troubleshooting topic might link to:

- Falling back to a prior release of an XYZ server
- Running the REORGANIZE DATA utility
- Resolving inconsistent data after a power failure
- Analyzing the XYZ trace table

A troubleshooting topic might include a <tsDiagnose> element that helps users determine which of two resolution procedures to follow, depending on what they discover by following the instructions in the <tsDiagnose> information. Assume that the primary resolution procedure, which is expected to be useful in the vast majority of cases, never needs to be reused, in which case you should use either the first approach (including the information in paragraph format) or the last approach (directly imbedding a task topic). Suppose, though, that the secondary, less-frequently used resolution procedure is a long and involved list of steps that is included in another plug-in in your information center. In this case, linking to the second resolution procedure is probably the best solution so that the troubleshooting topic that you are developing is not too long.

Imbed a task topic directly

The last approach to including task information is appropriate when the information needs the structure that a task topic requires (that is, more than just a simple paragraph), but when that task information does not exist elsewhere and is unlikely to ever be reusable in another context. When these criteria are met, this approach is best.

This approach is not appropriate if the information:

- Can be adequately explained in a simple paragraph, in which case the first approach is better.
- Is or might potentially be reusable, in which case the second (conref) or third (linking) approaches are better choices.

You can develop the task information to imbed in the troubleshooting topic in either of two ways:

- Locate the </tsBody> tag in the troubleshooting topic and select **Insert Markup** --> **Task** . The tagging and structure of a task topic is added to the troubleshooting topic, and you can put the information into the proper places in that element.
- Develop the task information outside of the troubleshooting topic by creating a new task topic. When the information is complete, you can copy and paste the content into the troubleshooting topic, immediately after the </tsBody> tag.

Whenever you use the approach that involves directly imbedding a task topic, the respective <tsDiagnose> or <tsResolve> element is omitted from the topic. For example, if the task topic that you imbed helps users diagnose the problem, you would not use the <tsDiagnose> element in the topic. Likewise, if the task topic that you imbed helps users resolve the problem, you would omit the <tsResolve> element from your topic.

If you imbed a task in place of the <tsDiagnose> element, do not use a <tsResolve> element. In place of the <tsResolve> element, you can either imbed a task or include a conref to a reusable object. For example, you might have this sort of structure:

<tsTroubleshooting>
<title></title>
<shortdesc></shortdesc>
<tsBody>
<tsSymptoms></tsSymptoms>
<tsCauses></tsCauses>
<tsEnvironment></tsEnvironment>
</tsBody>
imbedded task topic instead of <tsDiagnose> element
Either an imbedded task topic *or* a conref to reusable object in place of <tsResolve>
</tsTroubleshooting>

When you imbed a task topic in place of <tsDiagnose> or <tsResolve>, the related links for the troubleshooting topic are generated in the output immediately prior to the first reusable object.

Example of imbedding a task topic directly into a troubleshooting topic:

The following example shows some resolution information that might not be reusable and that does not fit well into a paragraph structure.

Figure3. Example of imbedding a task in a troubleshooting topic

```
<tsTroubleshooting xml:lang="en=us">
.
.
.
<tsBody>
.
.
.
</tsBody>
<task>
<title>Correcting the definition of a text-search server</title>
<shortdesc>The text-search server must be correctly defined if the XYZ
server is to
operate successfully. Incorrect text-search definitions can result in XYZ
failures.</shortdesc>
<taskbody>
<prereq>Identify the correct name for the text-search server that you
should use.
If you received an xyz1234E message, it identifies an incorrect name. If
you do not
know the correct name, contact your system administrator for this
information.</prereq>
<context>To correct the definition of a text-search server:</context>
<steps>
<step><cmd>Click the <uicontrol>XYZ server</uicontrol> tab to display the
XYZ
Server page.</cmd></step>
<step><cmd>In the <uicontrol>Server name</uicontrol> field, type the
correct
name of the XYZ server to associate with the text search.</cmd></step>
<step><cmd>Click <uicontrol>Test Connection</uicontrol> to test whether
you can
successfully connect to the server. A message window opens to inform you
of
the status.
If you cannot successfully connect, contact your system
administrator. </cmd></step>
<step><cmd>Click <uicontrol>OK</uicontrol> to save this server definition
and close
the window.</cmd></step>
```

```
</steps>
</taskbody>
</task>
</tsTroubleshooting>
```

Links in troubleshooting information

Links in troubleshooting topics are generally similar to links in other types of topics, with a few exceptions.

Although troubleshooting topic links are similar to links in other types of topics, two important differences exist:

- When you link from one troubleshooting topic to another troubleshooting topic, the output displays a link to the troubleshooting topic at the bottom of the topic under the heading “Related troubleshooting information.” (This behavior applies to relationship-table links and to the related-link element. Relationship-table links are highly recommended over related-link elements.)

  However, links from other types of topics to a troubleshooting topic appear under the “Related information” heading. (This heading behavior is a temporary issue with specializations in general, not just with the DITA Troubleshooting specialization. Eventually, all links to troubleshooting topics are expected to be included under the “Related troubleshooting information” heading.)

- In-line links are generally discouraged inside of the body of topics because they take the user away from the information in the topic to other information. Because of the nature of troubleshooting topics, the need for in-line links is more common than in other types of topics, including task topics. For example, users might be following a procedure, checking for different characteristics of the problem that they are troubleshooting, and at one point they might find that the current procedure is not the one that they need to use. In these cases, in-line links are not only acceptable but necessary so that the user finds the appropriate information for the specific problem. In-line links in troubleshooting topics are designed to take the user to another topic, and the user is generally not expected to return to the source of the link.

Information about links is covered in the DITA usage information.

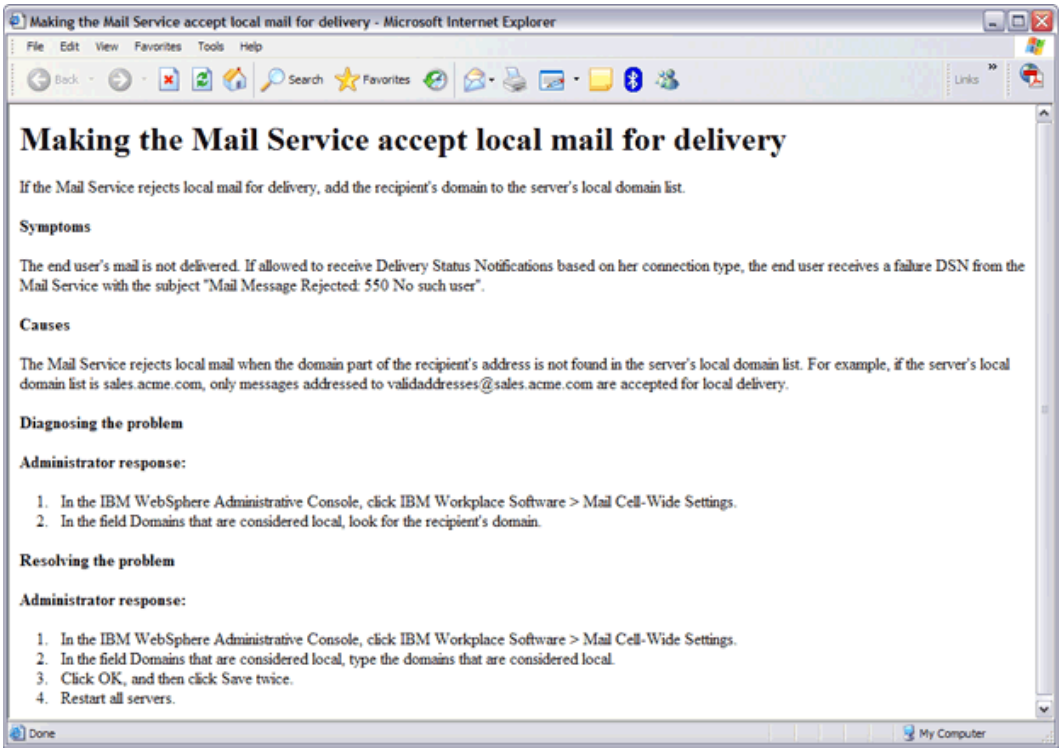
Troubleshooting topic outputs

Troubleshooting topics can be delivered in several different output formats.

You have multiple options for delivering troubleshooting topics. However, the initial, primary delivery vehicle is an Eclipse information center. As with other DITA topics, you can also choose to package your troubleshooting topics in a PDF. In the future, DITA topics might be able to be delivered in technotes.

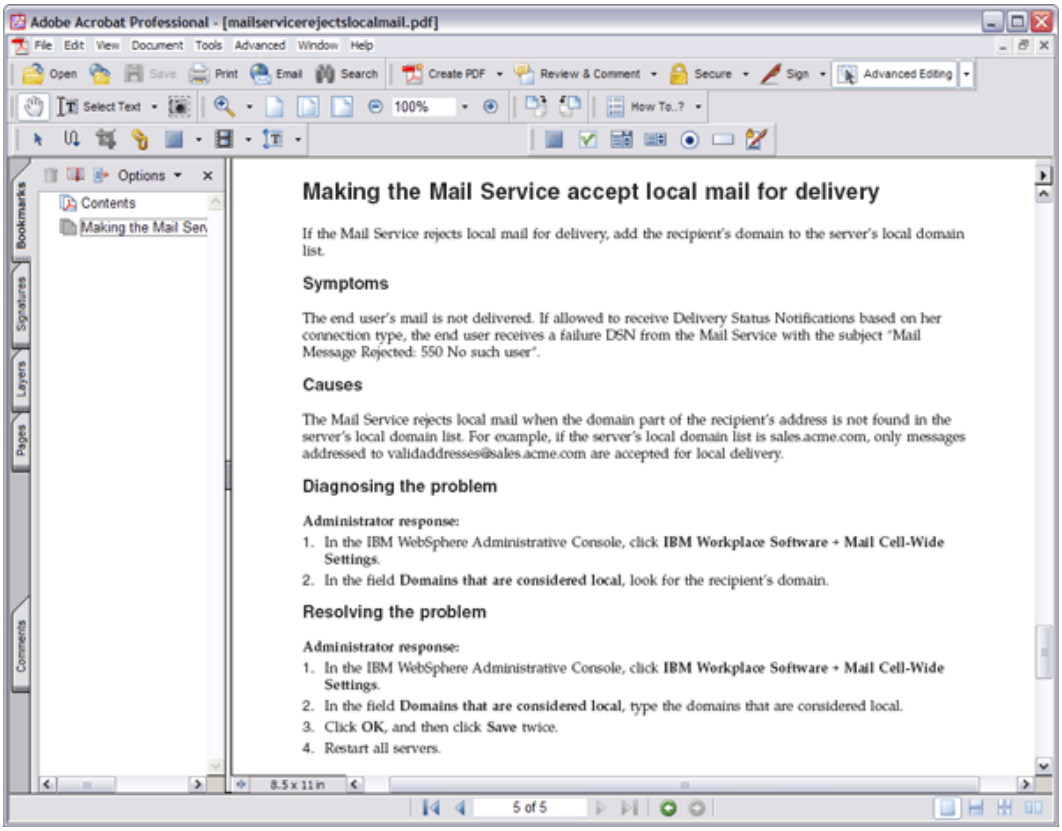
The following figure shows a sample troubleshooting topic in HTML format.

Figure4. Sample troubleshooting topic in HTML.



The following figure shows a sample troubleshooting topic in an online PDF.

Figure5. Sample troubleshooting topic in PDF.


Adobe Acrobat Professional - [mailservicerejectslocalmail.pdf]
File Edit View Document Tools Advanced Window Help
Open Save Print Email Search Create PDF Review & Comment Secure Sign Advanced Editing
Select Text 100% How To..?
Options
Contents
Making the Mail Ser
Bookmarks
Signatures
Layers
Pages
Comments
Making the Mail Service accept local mail for delivery
If the Mail Service rejects local mail for delivery, add the recipient's domain to the server's local domain list.
Symptoms
The end user's mail is not delivered. If allowed to receive Delivery Status Notifications based on her connection type, the end user receives a failure DSN from the Mail Service with the subject "Mail Message Rejected: 550 No such user".
Causes
The Mail Service rejects local mail when the domain part of the recipient's address is not found in the server's local domain list. For example, if the server's local domain list is sales.acme.com, only messages addressed to validaddresses@sales.acme.com are accepted for local delivery.
Diagnosing the problem
Administrator response:
1. In the IBM WebSphere Administrative Console, click IBM Workplace Software + Mail Cell-Wide Settings.
2. In the field Domains that are considered local, look for the recipient's domain.
Resolving the problem
Administrator response:
1. In the IBM WebSphere Administrative Console, click IBM Workplace Software + Mail Cell-Wide Settings.
2. In the field Domains that are considered local, type the domains that are considered local.
3. Click OK, and then click Save twice.
4. Restart all servers.
8.5 x 11 in
5 of 5

Migration tips

If you find that your existing troubleshooting information is not in the same format as the DITA troubleshooting topic, you need to assess the information and decide what to do with the text. A variety of tips can help you migrate existing information to the DITA troubleshooting topic.

Follow these tips when migrating existing information to the DITA troubleshooting topic type.

Embrace the new format.
The DITA troubleshooting specialization was designed by a cross-brand group of subject-matter experts who were asked to define the architecture of the ideal problem-solution-oriented troubleshooting topic. In addition, the specialization was validated by customers in multiple customer-feedback sessions. Using a single topic type for troubleshooting information provides many benefits, the most important of which is consistency for users. (Customer feedback on a prototype of this specialization affirmed that consistency in structure is valuable to users.)

Choose wisely which documents to migrate.
Some existing troubleshooting information might be more appropriate in a technote than in an information center topic. Evaluate existing information against criteria for technotes versus information center topics, and make conscious decisions about which topics to migrate to DITA troubleshooting topics in an information center.

Prioritize.
If you have a large volume of troubleshooting information, you might not have enough time to migrate it all to the DITA troubleshooting topic type immediately. If you have time constraints and cannot migrate all existing troubleshooting information to DITA troubleshooting topics, here are some ideas for how to prioritize which information to migrate first:

- Identify the specific problems that are most critical for users. Migrate the troubleshooting information for those problems first. During the next release cycle, you can migrate the less-critical troubleshooting information.
- Identify the parts of the product for which customers report the most problems. Migrate the troubleshooting information for those parts of the product first. (For example, if customer satisfaction is lowest for the installation task and users report many problems, migrate installation-related troubleshooting information initially.) This approach can provide the biggest "bang for the buck" because customer satisfaction can be positively affected in an area where it suffers. During the next release cycle, you can migrate the remaining troubleshooting information.
- Identify the troubleshooting information that will be easiest to migrate into the DITA troubleshooting topic, and migrate that information first. This is often called the “low-hanging fruit” strategy because you can accomplish this migration with fairly little time and resource. During the next release cycle, you can focus your efforts on researching and rewriting the more complicated troubleshooting information and migrating it into DITA troubleshooting topics at that time.

Chunk, when necessary.
If your existing troubleshooting information includes background information that is not absolutely critical to the description of the problem or its resolution, reorganize the information so that the background information is in a separate concept topic; the troubleshooting topic can then link to the related concept, and vice-versa. Likewise, if your existing troubleshooting information is organized such that a single topic covers multiple problems and their associated solutions, reorganize the information so that each problem and its solution is in a single topic. If an existing topic includes diagnosis or resolution task information that applies to multiple problems, you can set up the topic as a reusable object that multiple troubleshooting topics can reuse (by doing a conref or a link).

Enhance, when necessary.

Some existing troubleshooting information provides a lot of problem-determination steps but falls short of telling the user how to resolve or work around the problem. These topics, as they are, are not good candidates for the DITA troubleshooting topic. If you can enhance the information to tell the user (in the <tsResolve> element) how to resolve or work around the problem, do so. If you cannot enhance the information in this way, either because of time or knowledge limitations, focus your energy on migrating information about problems whose resolutions are known.

Help retrievability.

Users who have problems and take the time to look for solutions are generally in a hurry. Help users find the relevant troubleshooting topic for their problem by writing titles, short descriptions, and abstracts that make the topics easy to find.

Plan carefully, one topic at a time.

For each existing troubleshooting topic, determine which element in the DITA troubleshooting topic should contain various pieces of the existing text. If your existing information contains text that doesn't fit neatly into one of the DITA topic elements, consider these specific recommendations:

- If your existing information includes a “system action” section, consider whether the text describes some environmental details that are relevant at the time of the problem (before, during, or after the problem occurred). If the “system action” section includes this sort of information, you can put it in the optional <tsEnvironment> element, which is intended to provide environmental details that are not necessarily appropriate for the title, short description, or abstract of the topic.
- If your existing information identifies a single symptom or a single cause of the problem, you might be concerned about the plural label on the <tsSymptoms> and <tsCauses> elements. In many cases, problems would have multiple symptoms and multiple potential causes, so we chose to use plural labels for these elements. Feedback from customers indicates that users are unlikely to be bothered by this singular-plural situation.
- If your existing information identifies multiple potential causes that are mutually exclusive, include that information in the <tsCauses> element, making clear the fact that the multiple causes are mutually exclusive. If users need to take some action to determine which cause relates to their specific problem instance, include the user actions in the optional <tsDiagnose> element. This is particularly important in cases where the resolution of the problem varies based on the specific cause. In this case, you can include multiple <tsResolve> elements to cover the different ways that users can resolve the problem.
- If your existing information does not include any information about the cause of the problem, try to find out what the cause is and provide the information. You can omit the optional <tsCauses> element in these circumstances:
  - You can find no one in Development or Support who can provide the information about cause.
  - You do not have time to find the cause before you need to publish the information. In this case, record the need to add this information at your next opportunity.
  - You can find information about the cause, but the information should not be externalized to users.
- If your existing information does not include a role-based label on any actions for users to take (that is, anyone who reads the topic can perform the necessary actions), you do not need to use any of the optional role-based response elements (such as the <tsProgrammerResponse> element).
- If your existing information includes a role-based label that is not supported by any of the supported role-based response elements, you can use the <tsResponseRole> and associated <tsResponseRoleLabel> and <tsResponseRoleAction> elements to customize a user-role label that is appropriate for your problem.
- If your existing information includes a checklist for users to follow, you can use a choice list in troubleshooting topics.

Use of choice lists in troubleshooting topics

A lot of troubleshooting information includes maintenance analysis procedures (MAPs), which are choice lists or checklists that include yes-no questions to walk users through the task of diagnosing or resolving problems. The following recommendations can help you migrate existing MAP information to the DITA troubleshooting topic type.

Choice lists or yes-no checklists are beyond the scope of the DITA troubleshooting specialization. Consider these best practice recommendations:

- If you use yes-no branches in existing troubleshooting information, consider always starting your choice with "yes" and then ending with "no" for consistency.
- Consider putting the information in a choice table when appropriate.

Creating a choice list

Example of a choice list in a troubleshooting topic:

1. Are you working on a model 7037-A50 or 7047-185?
   - **Yes**: Go to Diagnosing a problem with a 7037-A50 or 7047-185.
   - **No**: Continue with the next step.
2. Are you troubleshooting a problem with the keyboard, video, and mouse (KVM) switch for the 1x8 or 2x8 console manager?
   - **Yes**: Go to Troubleshooting the keyboard, video, and mouse (KVM) switch for the 1x8 and 2x8 console manager.
   - **No**: Continue with the next step.
3. Is there an HMC attached to the failing server?
   - **Yes**: Continue with step 5.
   - **No**: Continue with the next step.
4. Is your system being managed by the Integrated Virtualization Manager?
   - **Yes**: Continue with step 13.
   - **No**: Refer to the appropriate procedure:
     - If you are having a problem with an AIX® server, go to AIX problem analysis.
     - If you are having a problem with an i5/OS® server, go to IBM i5/OS problem analysis.
     - If you are having a problem with a Linux® server, go to Linux problem analysis.

Example of a choice table in a troubleshooting topic

Instead of having yes-no branches in a MAP format, you might want to use a choice table.

Table2. Example choice table

Your situation	Action to take
You are working on a model 7037-A50 or 7047-185.	Go to "Diagnosing a problem with a 7037-A50 or 7047-185".
You are troubleshooting a problem with the keyboard, video, and mouse (KVM) switch for the 1x8 or 2x8 console manager?	Go to "Troubleshooting the keyboard, video, and mouse (KVM) switch for the 1x8 and 2x8 console manager".
Your system is managed by an HMC.	Continue with step 5.
Your system is managed by the Integrated Virtualization Manager (IVM).	Continue with step 13.
Your system is running AIX (and is not managed by an HMC or IVM).	Go to "AIX problem analysis".
Your system is running i5/OS (and is not managed by an HMC).	Go to "i5/OS problem analysis".

Your situation	Action to take
Your system is running Linux (and is not managed by an HMC or IVM).	Go to "Linux problem analysis".

Simple migration example

Sometimes information that you want to migrate to the DITA troubleshooting topic type is fairly straightforward, as this example shows.

The passage below is a section of existing information in a library that contains information that can fairly easily be migrated into the DITA troubleshooting topic.

Figure6. Simple example of troubleshooting informatio before migration to DITA

```
<d>
<dprolog><titleblk><title>XYZlocker failure
recovery</title></titleblk></dprolog>
<dbody>
<p><ph style="bold">Problem:</ph> XYZlocker fails in a wait, loop, or
abend.
</p>
<p><ph style="bold">Symptom:</ph> XYZlocker abends and the following
message
 appears:</p>
<xmp>XYZ122E <pv>zname</pv> ABEND UNDER XYZLOCKER IN MODULE
<pv>xxxxxxxx</pv> ABEND CODE <pv>
zzzz</pv>
</xmp>
<p><ph style="bold">System action:</ph> If XYZlocker abends, XYZ
terminates.
If XYZlocker waits or loops, terminate XYZlocker, and DB2 terminates
automatically.
</p>
<p><ph style="bold">System programmer action:</ph> None.</p>
<p><ph style="bold">Operator action:</ph> <ul>
<li>Start the XYZlocker if you did not set it for automatic start when
you
installed XYZ.</li>
<li>Issue the command START XYZ <pv>xyzid&lt;</pv>.</li>
<li>Start DB2</li>
</ul></p>
</dbody></d>
```

To migrate this passage into a DITA troubleshooting topic, you could take following actions:

1. Open a new DITA troubleshooting topic.
2. Add a title; if appropriate, consider using a task-oriented heading, such as “Recovering from XYZlock failure.”
3. Copy and paste the information in the “problem” section of the passage into the short description of the DITA topic.
   Tip: Do not automatically use the existing text exactly as it is. In this case, the text is a complete sentence, but sometimes it won't be. If that is true for you, rewrite the information so that it is in complete sentences. Also, if you think expanding the text can improve the short description, expand it accordingly.
4. Copy and paste the information in the “symptom” section into the <tsSymptoms> element.
5. Add a <tsCauses> element, and write text that explains the cause. This might require that you do some research and talk to subject-matter experts in Support or Development.
6. Because the DITA topic does not contain an element for a “system action” section, decide what to do with the text in that section. In this case, the “system action” section contains an imperative (“terminate XYZlocker”), which doesn't really fit into a section that explains what is going on with the system. You could use the <tsEnvironment> element to indicate: “If XYZlocker abends, XYZ terminates automatically. If XYZlocker waits or loops, XYZ does not automatically terminate.” Then you could put the imperative in the <tsResolve> element, which is intended for user actions.
   Tip: If your existing information includes a “system action” section, you might be able to place some or all of that information in the short description.

7. Ignore the “system programmer action” section because it says "None."
8. For the “operator action” section, decide which format is best.
    - If the current format (unordered list) is appropriate, use the <tsResolve> element. Then use the optional <tsOperatorResponse> element, and paste the unordered list of actions into the <tsOperatorResponse> element.
    - If you know that the information really should be in an ordered list format, do not use the <tsResolve> item. Instead, add a task after the </tsBody> tag, and include the information in task format in that location of the troubleshooting topic.
9. As usual, proof your information, and, if possible, have it edited.

The following DITA topic contains the revised troubleshooting information.

Figure7. Simple example of troubleshooting information after migration to DITA

```
<tsTroubleshooting>
<title>Recovering from XYZlocker failure</title>
<shortdesc>XYZlocker fails in a wait, loop, or abend.</shortdesc>
<tsBody>
<tsSymptoms>XYZlocker abends and the following message appears:
<codeblock>XYZ122E <varname>xyzname</varname> ABEND UNDER XYZLOCKER IN
MODULE
<varname>xxxxxxxx</varname>
ABEND CODE <varname>zzzz</varname></codeblock>
</tsSymptoms>
<tsCauses>NEED TO FIND OUT THE CAUSE!</tsCauses>
<tsEnvironment>If XYZlocker abends, XYZ terminates automatically. If
XYZlocker waits or loops, XYZ does not automatically
terminate.</tsEnvironment>
<tsResolve>
<tsOperatorResponse>
<ul>
<li>Start the XYZlocker if you did not set it for automatic start when
you installed
XYZ.</li>
<li>Start DB2.</li>
<li>Issue the command START XYZ <varname>xyzid</varname>.</li>
</ul>
</tsOperatorResponse>
</tsResolve>
</tsBody>
</tsTroubleshooting>
```

Complex migration example

Migrating existing troubleshooting information to the DITA topic type can sometimes be quite complicated. However, if the information is about a single problem for which resolution information is provided, you can analyze the existing information, and plan carefully how to move the information. This example shows a complex migration example.

The passage below is a section from an existing library that contains information that can be migrated into the DITA troubleshooting topic, but the migration is somewhat complex.

Figure8. Complex example of troubleshooting information before migration to DITA

```
<d>
<dprolog><titleblk><title>Application error
recovery</title></titleblk></dprolog>
<dbody>
<p><ph style="bold">Problem:</ph> An application program placed a
logically
 incorrect value in a table.</p>
<p><ph style="bold">Symptom:</ph>  SQL SELECT returns unexpected
data.</p>
<p><ph style="bold">System action:</ph> The system returns SQLCODE 0 for
the
SELECT statement because the error was not in SQL or DB2, but it was in
the
application program. That error can be identified and corrected, but the
data
in the table is now inaccurate.</p>
<p><ph style="bold">System programmer action:</ph> You might be able to
use
 RECOVER TOLOGPOINT to restore the database to a point before the error
occurred,
but  in many circumstances, you must manually back out the changes
introduced
by the  application. Among those circumstances are: <ul>
<li>Other applications changed the database after the error occurred.  If
you recover the table spaces modified by the bad application, you would
lose
all subsequent changes made by the other applications.</li>
<li>DB2 checkpoints were taken after the error occurred. In this case,
you
can use RECOVER TOLOGPOINT to restore the data up to the last checkpoint
before
the error occurred, but all subsequent changes to the database are
lost.</li>
</ul></p>
<p><ph style="bold">Procedure 1: If you have established a quiesce
point</ph> <ol>
<li>Run the REPORT utility twice, once using the RECOVERY option and once
using the TABLESPACESET option. On each run, specify the table space
containing
the inaccurate data. If you want to recover to the last quiesce point,
specify
the option CURRENT when running REPORT RECOVERY.</li>
<li>Examine the REPORT output to determine the RBA of the quiesce
point.</li>
<li>Execute RECOVER TOLOGPOINT with the RBA that you found, specifying
the
names of all related table spaces. Recovering all related table spaces to
the same quiesce point prevents violations of referential
constraints.</li>
</ol></p>
<p><ph style="bold">Procedure 2: If you have not established a quiesce
point
</ph> </p>
<p>If you use this procedure, you will lose any updates to the database
that
occurred after the last checkpoint before the application error
occurred.<ol>
<li>Run the DSN1LOGP stand-alone utility on the log scope that is
available
at DB2 restart, using the SUMMARY(ONLY) option.</li>
<li>Determine the RBA of the most recent checkpoint before the first bad
update
occurred, from message DSNR003I on the operator's console.</li>
<li>Run DSN1LOGP again using SUMMARY(ONLY) and specify the checkpoint RBA
as  the value of RBASTART. The output lists the work in the recovery log,
including  the information about the most recent complete checkpoint, a
```

```
summary
of all processing occurring, and an identification of the databases that
are
affecgted by each active user.</li>
<li>One of the messages in the output (identified as DSN1151I or
DSN1162I)
describes the unit of recovery in which the error was made. To find the
unit
of recovery, use your knowledge of the time the program was run (START
DATE=
and TIME=), the connection ID (CONNID=), authorization ID (AUTHID=), and
plan
name (PLAN=). In that message, find the starting RBA as the value of
START=.
</li>
<li>Execute RECOVER TOLOGPOINT with the starting RBA you found in the
previous
 step.</li>
<li>Recover any related table spaces or indexes to the same point in
time.
</li>
</ol></p>
<p><ph style="bold">Operator action:</ph> None.</p>
</dbody></d>
```

If you were to migrate this information to the DITA troubleshooting topic, you might take these steps:

1. Open a new DITA troubleshooting topic.
2. Add a title, but consider the best practice of using a task-oriented heading, such as “Recovering from an application error.”
3. Copy and paste the information in the “problem” section of the passage into the short description of the DITA topic.
4. Copy and paste the information in the “symptom” section into the <tsSymptoms> element.
5. Because the information in the “system action” section really describes the cause, copy and paste that information into the <tsCauses> element.
6. Add a <tsResolve> element and a <tsSystemProgrammerResponse> element.
7. Copy and paste the text of the “system programmer action” section into the <tsSystemProgrammerResponse> element.
8. After the </tsBody> tag, insert two tasks, one to hold the first procedure and one to hold the second procedure. However, give these two parallel, subordinate tasks better titles that communicate the user task rather than just “Procedure 1” and “Procedure 2.” Possible titles are “Recovering data from an established quiesce point” and “Recovering data without a quiesce point.”
9. Add some text to the bottom of the <tsResolve> element to indicate that the system programmer should use whichever procedure is appropriate, depending on whether a quiesce point is established.
10. Revise step 4 of the second procedure so that it begins with an imperative.
11. At the end of the second procedure, omit the operator response because it says “None.”
12. As usual, proof your information, and, if possible, have it edited.

The following DITA topic contains the revised troubleshooting information.

```
<tsTroubleshooting>
<title>Recovering from an application failure</title>
<shortdesc>An application program placed a logically
incorrect value in a table.</shortdesc>
<tsBody>
<tsSymptoms>SQL SELECT returns unexpected data.</tsSymptoms>
<tsCauses>The system returns SQLCODE 0 for the SELECT statement because
the error
was not in SQL or DB2, but it was in the application program. That error
can be
identified and corrected, but the data in the table is now
inaccurate.</tsCauses>
<tsResolve><tsSystemProgrammerResponse>
You might be able to use RECOVER TOLOGPOINT to restore the database to a
point before the error occurred, but in many circumstances, you must
manually
```

```
back out the changes introduced by the application. Among those
circumstances are:
<ul>
<li>Other applications changed the database after the error occurred.  If
you recover the table spaces modified by the bad application, you would
lose
all subsequent changes made by the other applications.</li>
<li>DB2 checkpoints were taken after the error occurred. In this case,
you
can use RECOVER TOLOGPOINT to restore the data up to the last checkpoint
before
the error occurred, but all subsequent changes to the database are
lost.</li>
</ul>
<p>The remaining response depends on whether a quiesce point has been
established.
Follow the appropriate procedure.</p>
</tsSystemProgrammerResponse>
</tsResolve>
</tsBody>
<task>Recovering data from an established quiesce point
<shortdesc>The best way to recover data is by relying on an established
quiesce
point because this approach results in minimal or no data
loss.</shortdesc>
<context>To recover data from an established quiesce point:</context>
<taskbody>
<steps>
<step><cmd>Run the REPORT utility twice, once using the RECOVERY option
and once
using the TABLESPACESET option. On each run, specify the table space
containing
the inaccurate data. If you want to recover to the last quiesce point,
specify
the option CURRENT when running REPORT RECOVERY.</cmd></step>
<step><cmd>Examine the REPORT output to determine the RBA of the quiesce
point.</cmd>
</step>
<step><cmd>Execute RECOVER TOLOGPOINT with the RBA that you found,
specifying the
names of all related table spaces. Recovering all related table spaces to
the same quiesce point prevents violations of referential
constraints.</cmd></step>
</steps>
</taskbody>
</task>
<task>Recovering data without a quiesce point
<shortdesc>If you use this procedure, you will lose any updates to the
database that occurred after the last checkpoint before the application
error
occurred.</shortdesc>
<context>To recover data without a quiesce point:</context>
<steps>
<step><cmd>Run the DSN1LOGP stand-alone utility on the log scope that is
available at
DB2 restart, using the SUMMARY(ONLY) option. </cmd></step>
<step><cmd>Determine the RBA of the most recent checkpoint before the
first
bad update
occurred, from message DSNR003I on the operator's console.</cmd></step>
<step><cmd>Run DSN1LOGP again using SUMMARY(ONLY) and specify the
checkpoint RBA as
the value of RBASTART. The output lists the work in the recovery log,
including
the information about the
most recent complete checkpoint, a summary of all processing occurring,
and an
identification of the databases that are affecgted by each active
user.</cmd></step>
<step><cmd>One of the messages in the output (identified as DSN1151I or
DSN1162I)
describes the unit of recovery in which the error was made. To find the
unit
of recovery, use your knowledge of the time the program was run (START
DATE=
and TIME=), the connection ID (CONNID=), authorization ID (AUTHID=), and
plan
name (PLAN=). In that message, find the starting RBA as the value of
START=.</cmd></step>
<step><cmd>Execute RECOVER TOLOGPOINT with the starting RBA you found in
the previous
step.</cmd></step>
<step><cmd>Recover any related table spaces or indexes to the same point
in
time.</cmd></step>
</steps>
```

```
</task>
</tsTroubleshooting>
```

Figure9. Complex example of troubleshooting information after migration to DITA

Troubleshooting topics as containers

When a set of related troubleshooting topics is collected in a container, or if the overall purpose of a set of topics supports the troubleshooting task, use a troubleshooting topic for the container topic.

When you create a container topic that is a troubleshooting topic type, you need at least a <shortdesc> element and a <tsSymptoms> element. You can also include other elements if you have information to include that applies to the entire set of topics in that container.

In the <tsSymptoms> element, describe the range of symptoms, if possible, or, if that is not possible, indicate that the symptoms vary based on the specific problems. </tsSymptoms>

Depending on what text is in the last element that you include in the container topic, you might want to include some additional content that provides a transition between the container topic content and the child-topic links that are generated in the output.

For example, assume that your last element is a <tsEnvironment> element that indicates "In most situations, the connection between product X and product Y remains active, but if utility work was interrupted by the failure, the connection might fail." If the output goes from the "Environment" label with this sentence immediately into the child links, users might think that the links are part of the <tsEnvironment> element, which is not the case. If the last content in your topic might leave users with any confusion, consider one of the following approaches:

- Add a <tsResolve> element, which you normally would not have in a troubleshooting container topic, and have the text within that element read something like: "Use the procedure that best matches the characteristics of your problem." End the sentence with a period rather than a colon.
- Add a new paragraph to the bottom of one of the elements (such as <tsSymptoms>, <tsCauses>, or <tsEnvironment>) that provides a subtle but useful transition into the child links. For example: "The approach that you take to resolve this type of problem depends on the particular characteristics of the problem." End the sentence with a period rather than a colon.

Placement of troubleshooting topics in the navigation structure

When you deliver troubleshooting topics in an information center, you have several options for where to place the topics in the navigation structure and how to organize groups of related troubleshooting topics.

Where you put troubleshooting topics in the navigation structure for your information center depends on several factors:

- The number of troubleshooting topics
- The number of tasks that the troubleshooting topics support
- Whether you also deliver PDFs that your users depend on
- How you expect most users to look for the troubleshooting information

No strict guidelines exist for the decision about where to place troubleshooting topics, but here are a few options for you to consider:

Use the "Troubleshooting" node.
If your Information Center has a problem determination node or troubleshooting node, this node is a reasonable place to deliver troubleshooting topics. If you have a small number of troubleshooting topics, you can place them directly under the node itself. If you have a larger number of troubleshooting topics that fit in different categories, you can have containers for the troubleshooting topics for the different categories, and the topics themselves can be subordinate to those containers. For example, if you have troubleshooting topics for the tasks of administration, installation, and application development, you would have the standard troubleshooting at the top of the "Troubleshooting" node. Next you could have a container for "Troubleshooting installation problems," "Troubleshooting administration problems," and "Troubleshooting application development problems." Work with your team's information architect to ensure that your navigation titles conform to the guidelines for your information center.

Use the appropriate task-specific node for troubleshooting topics.
If you want, you can place the troubleshooting topics in the task-specific nodes to which they apply instead of in the "Troubleshooting" node. For example, troubleshooting topics that support the installation tasks can be in the "Installing" node, troubleshooting topics that support the administration task can be in the "Administering" node, and so on. When you include troubleshooting topics in a task-specific node, generally place them toward the end of that cluster of topics.

Use one approach for the information center and another for the PDF.
If you produce PDFs for your users because they regularly use them, you can place the troubleshooting topics in the "Troubleshooting" node of the information center, but you can then adjust the ditamap for the PDF of the task-based information so that the troubleshooting topics are pulled in with the tasks that they support. For example, if you have a large number of administration-related troubleshooting topics, you can:

- Include the topics in the *Administration Guide* that you provide to users.
- Include the topics in the "Troubleshooting" node of the information center rather than the "Administration" node.

Place critical troubleshooting topics in the task-specific node and in the "Troubleshooting" node.
Certain troubleshooting topics might be important enough to be included in multiple places in the navigation structure. For example, a topic that gives useful guidance on resolving a critical installation problem might deserve to be placed in the "Installing" node and in the "Troubleshooting" node. Be aware, however, that duplicating a large number of troubleshooting topics (or topics of any type) in the navigation structure results in a longer overall structure, from top to bottom, which can complicate the task of maintaining the navigation structure and also might slow down users in finding what they are looking for. Therefore, you might want to reserve this approach for only the most critical topics.

Place the troubleshooting topics for the tasks that they support in the task-specific plug-ins but provide links to them from the "Troubleshooting" node.

This approach offers the benefit of keeping the troubleshooting topics with the other topics for the associated task (in terms of navigation structure and file maintenance), but it also makes their existence visible to and retrievable by users who go directly to the "Troubleshooting" node. If you choose this approach, you can include a topic in the "Troubleshooting" node called "Troubleshooting problems with other tasks." That topic can then provide links to the various container topics for troubleshooting topics that exist in different task-specific nodes in the information center.

When you have a large number of troubleshooting topics, and you want to organize some or all of them into logically related groups, you might wonder what type of topic to use for the container. In this case, you can use a troubleshooting topic type for the container, and associate the parent-child relationships between containers and subordinate troubleshooting topics in the ditamap.

Troubleshooting topic elements

A troubleshooting topic, which is derived from the generic topic type, explains what actions users can take to resolve or work around a problem. Use the elements of the troubleshooting topic to describe the problem, to identify symptoms, causes, and environmental characteristics, and to tell users what to do to resolve the problem.

A troubleshooting topic can:

- Describe the general problem.
- Identify the symptoms, causes, and environmental details of the problem.
- Provide diagnosis steps.
- Explain the steps that users can take to resolve the problem. The troubleshooting topic can provide either a permanent resolution to the problem or a temporary workaround that helps users until the problem is permanently fixed in a future release of the product.

The troubleshooting topic might include the actual diagnosis and resolution task information, it might imbed task topics, or it might link to topics that exist elsewhere.

Role-based response elements

The optional role-based response elements are useful when the specific actions that are to be taken in a troubleshooting task topic or section are to be performed primarily by a certain type of user, such as an operator or network administrator.

These elements are part of the DITA troubleshooting specialization and are derived from the paragraph element. The following table identifies the available role-based response elements and their corresponding, predefined labels.

Table3. Predefined user response role elements and labels

Troubleshooting element	Label that is displayed
<tsAdministratorResponse>	Administrator response:
<tsApplicationProgrammerResponse>	Application programmer response:
<tsDatabaseAdministratorResponse>	Database administrator response:
<tsHardwareServiceProviderResponse>	Hardware service provider response:
<tsNetworkAdministratorResponse>	Network administrator response:
<tsOperatorResponse>	Operator response:
<tsProgrammerResponse>	Programmer response:
<tsSecurityAdministratorResponse>	Security administrator response:
<tsSoftwareServiceProviderResponse>	Software service provider response:
<tsSystemAdministratorResponse>	System administrator response:
<tsSystemProgrammerResponse>	System programmer response:
<tsUserResponse>	User response:

Each label, followed by a colon, is generated and displayed in your output. Text that you include after the response element start tag is displayed on the same line as the label, unless the text is in an element that starts on a new line (such as <ul> or <ol>).

For situations in which the predefined role-based response elements do not include the type of user who needs to perform a troubleshooting task, use the <tsResponseRole> element, and customize a label that works for the situation.

Sample code

```
<tsResolve>
<tsOperatorResponse>
<ol>
<li>Assure that no I/O requests for the failing disk are incomplete by
forcing the
volume offline by issuing the following command, replacing
<varname>xxx</varname> with
the unit address: <p><userinput>FORCE
<varname>xxx</varname>,OFFLINE</userinput></p>
</li>
<li>Issue the following command to stop all databases and table spaces
that reside
on the affected volume, specifying the names of the database and table
space.
<p><userinput>Stop dbase(<varname>database-name</varname>)
SPACENAM(<varname>space-name
</varname>)</userinput></p></li>
</ol>
</tsOperatorResponse>
<tsSystemProgrammerResponse>
Recover the affected table spaces by using the RECOVER utility.
</tsSystemProgrammerResponse>
</tsResolve>
```

PDF file

A PDF version of the documentation for the troubleshooting specialization is available online.

For a PDF version of this documentation, click here .