Guidelines for Contributing User Assistance to the DTP (Data Tools Platform) Project

v 1.0

Table of Contents

Versions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Emily Kapner, Maria Brownstein, Dave Resch	Initial Draft	2/28/07

Overview

This document outlines the recommended approach for contributing user assistance content to DTP. User assistance includes Welcome page content, help content, cheat sheets, and context-sensitive help files, usually delivered in the form of plug-ins. The purpose of these guidelines is to create an approach that is both consistent and scalable, and that allows the DTP project team to include or exclude help content contributions based on DTP capabilities (categories/activities).

Current DTP 1.0 Documentation Structure

The DTP 1.0 documentation was not architected for scalability. The current DTP 1.0 user documentation includes:

- a single plug-in containing help content for Connectivity and SQL Development topics,
- an intro plug-in that extends the SDK universal intro contributing DTP Welcome content,
- a plug-in for API documentation, and
- a context-sensitive help plug-in for DTP 1.0 BIRT that bridges DTP to ODA.

For future releases, it would be best to architect the user assistance contributions so that they align with DTP capabilities. This will allow the DTP project team to create different installation packages and only include the applicable set of corresponding help content-contributing plug-ins. This document identifies how to implement this architecture.

Project Architecture Decisions

Several project-level architecture decisions affect the design and implementation of user assistance plug-ins, particularly the division of content across multiple plug-ins, and the coordination of plug-in IDs.

These project-level architecture decisions must be made early in the release planning stage, and they must be communicated clearly to both Development teams and Documentation teams, ideally in the form of architectural specifications.

Note: Adequate architectural specifications are critical for the design and implementation of the user assistance contributions and to ensure the publications team can meet release schedules.

Capabilities

Capabilities (categories/activities) can be used to enable or disable the plug-ins that contribute functionality and/or UI components, effectively “filtering” the user interface to reduce complexity and enhance usability.

Capabilities can be switched on and off by:

- Associating a “trigger” with a user action (e.g., changing the focus in a view or perspective), so that capabilities can be switched programatically through user interaction
- Defining a preference page and implementing a class that populates that preference page, so that capabilities can be accessed directly by the user through the Preferences dialog

Note: The Preference dialog need not expose all capabilities to the user.

The project should design capabilities, and define their associations with features and plug-ins to provide appropriate UI filtering.

Capabilities should be contributed by UI plug-ins, so declaring and implementing capabilities is the responsibility of UI plug-in developers.

Component Hierarchy

Ultimately, the project architecture should be able to be expressed as a hierarchy of “components.” For example:

```
DTP project
        Installation Package A
                Feature 1
                        Plug-in 1.a
                        Plug-in 1.b
                        ...
                Feature 2
                        Plug-in 2.a
                        Plug-in 2.b
                        ...
                ...
        ...
```

This hierarchy will facilitate an appropriate feature and plug-in architecture for both project components and user assistance contributions.

Architectural Specifications

To facilitate user assistance architecture and design, the following questions must be answered by the project architectural specifications:

Download/Installation Packages

- Will the project provide multiple installation packages? (e.g., subsets of the complete project deliverables)
- Will any installation packages be defined at a level higher than features?
    - If so, how will the relationships between multiple features in an installation package be expressed? (e.g., dependency, colocation affinity, “nested” features, etc.)

Features

- What are the features to be delivered?
- What is the ID of each feature to be delivered?
- What functionality and UI components will be included in each feature?
- What is the ID of each plug-in that contributes the functionality and/or UI components in each feature?
- Will any features be "nested?" (e.g., features that include other features)
    - If so, how many top-level features will be delivered?
    - What is the feature ID of each top-level feature?
    - What are the dependencies between top-level features?
    - How many subfeatures will be included in each top-level feature?
    - What is the feature ID of each subfeature in each top-level feature?

Note: These questions may iterate to as many levels as features are nested.

Capabilities (Categories)/Activity Enablement

How will capabilities correlate to features?

- Which features will be associated with each capability (or activity)?

- How will capabilities correlate to the plug-ins that contribute functionality and/or UI components?
    - Which plug-ins will be associated with each capability (or activity)?

User Assistance Plug-in Structure

Each user assistance component should be delivered as a separate plug-in. These components include, but are not limited to, the following:

- Welcome page (intro) extensions
- Help content
- Context-sensitive help contributions
- Cheat sheets

Contributing different types of user assistance content in separate plug-ins allows more flexibility in defining the content architecture (i.e., plug-ins that contribute content at the appropriate level in the project component hierarchy).

The following sections cover guidelines for each of each type of user assistance component.

Documentation Feature IDs and Plug-in IDs

To take advantage of capabilities, documentation features and plug-ins should follow the naming conventions of their corresponding project features and UI plug-ins.

For example:

- Project feature ID **org.eclipse.datatools.feature**
- Documentation feature ID **org.eclipse.datatools.feature.doc**
- UI plug-in ID **org.eclipse.datatools.plugin**
- Documentation plug-in ID **org.eclipse.datatools.plugin.doc**

This allows both documentation components and their corresponding functional (or UI) components to be matched to a single activityPatternBinding that specifies “org.eclipse.datatools.feature*” or “org.eclipse.datatools.plugin*”.

Extending the example above, plug-ins that contribute different types of user assistance content should conform to the following ID conventions:

- Documentation (main help) content: org.eclipse.datatools.plugin.**doc**
- Welcome page (intro) content: org.eclipse.datatools.plugin.**intro**
- Context-sensitive help: org.eclipse.datatools.plugin.**contexts**
- Cheat sheets: org.eclipse.datatools.plugin.**cheatsheets**

Welcome Page (Intro)

Extensions to the universal intro should be contributed by one or more separate plug-ins, based on their relevance to project features and UI plug-ins.

Project-level Plug-in

All installation packages should include a project-level intro contribution plug-in, which points to the anchors defined by the universal intro, and declares additional anchors for lower-level intro plug-ins. Thus, the project-level intro plug-in contributes the “framework” for all DTP project intro content contributions.

By including the project-level plug-in in every installation package, we ensure that the project intro content will be presented consistently in the Welcome pages, regardless of which particular project subsets or features are installed (or enabled) at any time.

Help Content

Based on mapping of documentation plug-ins to project capabilities, each component of DTP should have a separate documentation plug-in.

For example, Connectivity should have a documentation plug-in, SQL Development should have a documentation plug-in, etc.

Project-level Plug-in

All installation packages should include a common, project-level documentation plug-in, which declares anchors (content extension points) for lower-level documentation plug-ins. Thus, the project-level plug-in contributes the “framework” for all DTP project documentation presented on the bookshelf (Help window TOC).

By including the project-level plug-in in every installation package, we ensure that the project documentation will be presented consistently in the help system, regardless of which particular project subsets or features are installed (or enabled) at any time.

TOC Contributions

All documentation plug-ins must use the **org.eclipse.help.toc** extension point in their plugin.xml files to declare TOC contributions.

Only the project-level documentation plug-in will declare its TOC contribution to be primary (i.e., appearing at the highest level in the Help window), so the TOC contributions of all lower-level plug-ins will appear only within the hierarchy of the project-level TOC.

The project-level documentation plug-in makes this declaration in the plugin.xml file:

```
<extension point="org.eclipse.help.toc">
   <toc file="toc.xml"
      primary="true"/>
   </extension>
```

All other documentation plug-ins must declare their TOC contributions as follows:

```
<extension point="org.eclipse.help.toc">
   <toc file="toc.xml"
      primary="false"/>
   </extension>
```

or

```
<extension point="org.eclipse.help.toc">
   <toc file="toc.xml"/>
   </extension>
```

Note: The default is `primary="false"`.

Anchors and Links in toc.xml

The project-level documentation plug-in will declare anchors in its toc.xml file, to which the lower-level documentation plug-ins will link.

When rendered by the Help system, the TOC contributions of lower-level plug-ins are merged into the project-level TOC. The anchor location that each lower-level plug-in links to will determine the location of its TOC contribution within the project-level TOC.

Note: If multiple TOC contributions link to the same anchor, the order in which the contributions are presented is indeterminate, and it could vary from time to time. Therefore, each anchor should be targeted by only one lower-level TOC contribution.

Anchors declared in the project-level plug-in's toc.xml file look like this:

```
<toc label="Data Tools Platform">
   <topic href="overview.html" label="Project-level Overview Content">
   <anchor id="feature1_contrib"/>
   <anchor id="feature2_contrib"/>
   ...
</toc>
```

A link is declared by using the *link_to* attribute on the <toc> element in the lower-level plug-in's toc.xml file:

```
<toc label="DTP Feature 1 Name"
   link_to="../org.eclipse.datatools.doc/toc.xml#feature1_contrib">
   <topic href="feature1_content.html" label="Feature 1 Content"/>
   ...
</toc>
```

DITA Maps

The DTP project-level map is created using DITA. In this ditamap, anchors will be defined using <anchor> elements:

```
<map id="org.eclipse.datatools.doc">
   <topicref href="some_DTP_content.xml"/>
   <topicref href="more_DTP_content.xml">
      <anchor id="feature1_contrib"/>
   </topicref>
   <anchor id="feature2_contrib"/>
   <anchor id="feature3_contrib">
      <anchor id="DTP_contrib_A"/>
      <anchor id="DTP_contrib_B"/>
   </anchor>
   <anchor id="feature4_contrib"/>
</map>
```

Note: In terms of the map structure, an <anchor> element is valid anywhere that a <topicref> element is. Also, as shown above, <anchor> elements can be nested the same way as <topicref> elements to create hierarchical structures.

If DITA is used for lower-level documentation plug-ins, links will be defined using the *anchorref* attribute on the <map> element in the ditamap:

```
<map id="org.eclipse.datatools.blahblah.doc"
   anchorref="../org.eclipse.datatools.doc/toc.xml#feature1_contrib">
   <topicref href="DTP_content.xml"/>
   ...
</map>
```

Note: The value of the *anchorref* attribute must be the plug-in-relative path, in the Eclipse installation, from the plug-in generated by the ditamap to the <anchor> element ID, in the toc.xml file of the project-level plug-in.

Maps that use *anchorref* can also use <anchor> elements to implement multi-tiered documentation plug-ins, which might be necessary to deliver content partitioned appropriately for the project's component architecture.

Context-sensitive Help

Context-sensitive help infrastructure (i.e., mapping context IDs to specific help topics) will be contributed by dedicated plug-ins. These plug-ins will contribute context manifests (contexts.xml files) that point to content contributed by other documentation plug-ins (the core help content), but they will not contribute any content themselves.

Context-sensitive Help Implementation

In the UI code, editors and other controls will implement methods of IContextProvider to handle context-sensitive help requests. Rather than using help context IDs directly, UI controls will use help keys declared as public static final String in an IHelp* class. Those keys will be used to look up the actual context IDs in a .properties file. For example:

```
public int getContextChangeMask() {
      return SELECTION;
}
public IContext getContext(Object target) {
      return HelpSystem.getContext(getContextId(getHelpKey(target)));
}
public String getSearchExpression(Object target) {
      return getSearchExpression(getHelpKey(target));
}
```

Each help key will be associated with two string values:

- The actual help context ID
- A Help search expression

Key-value pairs will be defined in two .properties files (one for context IDs and one for search expressions).

Though meaningless to Eclipse, help keys provide an abstraction from help context IDs in the UI code, with the following benefits:

- Simplified handling of help context IDs and search expressions within UI code.
- Development teams can associate new help keys with UI controls, without necessitating that corresponding help context IDs exist.
- Documentation teams can modify help context IDs, without necessitating a change in UI code.
- Generating the Doc-side infrastructure for context-sensitive help can be automated.

Context-sensitive help plug-ins will contribute:

- The context ID and search expression .properties files
- One or more context manifests (contexts.xml files)
- (Optionally) other utilities functions that may be required to support the context-sensitive help implementation

Dev procedure:
Development teams will be responsible to:

- Implement methods of IContextProvider for all appropriate UI controls, and
- Communicate a list of help keys to Documentation teams:
  - The list of help keys will be supplied in IHelp*.java files, which will be sent to Documentation as soon as they change, along with a diffs file that indicates any changes since the last time it was sent to Documentation.
  - These files will be a series of public static final String declarations with accompanying comments that provide a cue as to where to find the associated control in the UI.

Doc procedure:
Documentation teams will be responsible to:

- Maintain the context ID and search expression .properties files, based on help key information communicated from Development teams, and
- Update the context-sensitive help plug-ins that contribute those files. (This part can be automated.)

Plug-in Structure

Because a context-sensitive help plug-in will contribute properties files, which could be common to several UI plug-ins, as well as a context manifest that could support several UI plug-ins, the "component level" with which a context-sensitive help plug-in is associated must be evaluated carefully.

Note: The appropriate component level for a context-sensitive help plug-in may well be different than the component level(s) of its corresponding "content-contributing" documentation plug-ins.

Documentation teams must rely on the project's architectural specifications and the IHelp*.java (help key constants) files provided by Development teams to determine the appropriate capabilities and features with which to coordinate context-sensitive help plug-ins.

Cheat Sheets

Similar to documentation and context-sensitive help plug-ins, cheat sheets should be contributed by separate plug-ins so they can be activated/deactivated based on capabilities associated with UI plug-ins.

Cheat sheet content will be developed using the Cheat Sheet Editor provided in Eclipse 3.3.

Cheat sheet plug-ins could be constructed from a plug-in project in the Eclipse PDE.

Delivery Formats

All user assistance plug-ins will be delivered in features.

All user assistance plug-ins will be delivered as jar files, with an appropriate jar manifest.