ENOVIA ProjectSync
Advanced Customization Guide

3DEXPERIENCE 2022

Table Of Contents

ProjectSync Customizations

Introduction

ENOVIA Synchronicity ProjectSync® is a web-accessible database used to manage electronic design life-cycle data. Through a standard web browser, teams can share issue, bug, and engineering change order (ECO) data associated with design projects. Built within the DesignSync SyncServer architecture, ProjectSync:

- Is tightly integrated with DesignSync, letting teams access design configuration management and revision control data on their own web browsers.
- Enforces security using encryption, user authentication, and customizable access control policies.
- Supports email notification, as well as triggers and events so that you can develop powerful Tcl scripts to customize your ProjectSync environment.
- Helps globally dispersed design teams enforce design practices and methodologies by modeling design processes as ProjectSync notes.
- Provides a highly customizable user interface, letting you design your own applications such as project home pages and forecasting tools.

This Guide illustrates methods of customizing the ProjectSync interface so that you can tailor the user interface directly to your project team's needs.

Note: The ProjectSync programming interface may change and evolve with the product. Changes are documented as they occur.

Note on using this guide: References from the *ENOVIA ProjectSync Advanced Customization Guide* to the *ENOVIA Synchronicity Command Reference* guide always link to the ALL version of the guide, which contain information about all working methodologies for DesignSync. For more information about the available working methodologies, see ENOVIA Synchronicity Command Reference.

Types of ProjectSync Customizations

ProjectSync uses the abstraction of a **note** to help you model engineering processes. For example, you can create a SyncDefect or HW-Defect-1 note to submit a bug report. You can create a Note, the default ProjectSync note type, to start a discussion about a project. ProjectSync automatically emails the note entries to team members who have subscribed for that project and note type. Later you can query the note to view the history of the issue or defect. To understand the basics of notes and note types, see ProjectSync User's Guide: What Is a Note?

The term **panel** refers to a ProjectSync window. The most prevalent type of panel in ProjectSync is a **note panel**, a form most often used to gather information from a user.

ProjectSync lets you customize most aspects of the product using notes and panels. You can customize existing ProjectSync note types, create new note types, add new fields to existing note types using the Note Type Manager, accessible from the ProjectSync menu. You can customize the ProjectSync main menu, welcome page, and data sheets. You implement these customizations using the ProjectSync graphical interface. See ProjectSync Help to make these customizations.

For other more complex customizations, you write scripts and edit initialization files that correspond to ProjectSync panels. These types of customizations are covered in this document. Using this Guide, you can

- Change the appearance of panels, such as note type panels, by developing HTML code for the panel
- Change the behavior of existing note type panels by creating an associated panel initialization script
- Override the default ProjectSync behavior of commands and panels
- Specify new behavior for the gather phase of a note type -- the phase when the note collects input in its panel fields
- Specify new behavior in the process phase of a note type -- the phase when ProjectSync processes the note
- Generate a customized results panel
- Store alternate versions of a panel and display a specific version based on some criteria, such as whether the user belongs to a particular group of users

ProjectSync Panel Architecture

Anatomy of ProjectSync Note Types

The ProjectSync panels you will likely customize most often are note type panels. You work with these panels to create new note types or to modify existing note types. Whereas a note is a database record in the ProjectSync relational database, a note type defines the set of properties that make up the database record. Each note you create is associated with a particular note type.

When you create a note type, you specify the properties and their property types. A property corresponds to a field name. The property type defines the type of data the property represents. The property type not only signifies the type of data, but also determines the graphical element displayed within the note. The graphical elements displayed for particular property types can be text boxes, editable text fields, editable multi-line text boxes, read-only fields, pull-down fields, and calendar pop-ups. See ProjectSync User's Guide: Predefined Property Types for a complete list of property types.

The type of graphical element displayed for a property type depends also on the mode of the note type. Note types can have modes for Add, View, and Edit. For instance, the String property type, which corresponds to a multi-line text box, displays as an editable text box in Edit mode and a read-only text box in View mode.

Another distinction to keep in mind about note properties is that the property name does not display within the note panel. Instead, when you create a note type, you specify a prompt for each property that displays in the note panel.

A good way to become familiar with note types and properties is to select the Note Type Manager from the ProjectSync main menu and select **Modify an existing note type**. Select one of your installed note types such as SW-Defect-1. If you have not installed any note types, select **Install one of the prepackaged note types** and follow the prompts to install a note type. If you are installing a note type only to experiment with it, you can later remove it by selecting **Delete a note type**.

The Modify Note Type panel displays the field names (properties), the field types (property types), the prompts, and default values of each property. If you do not make any changes and then select the **Next>>** button, "No Action Required" displays. You can also experiment with modifying properties and then delete and reinstall the note type.

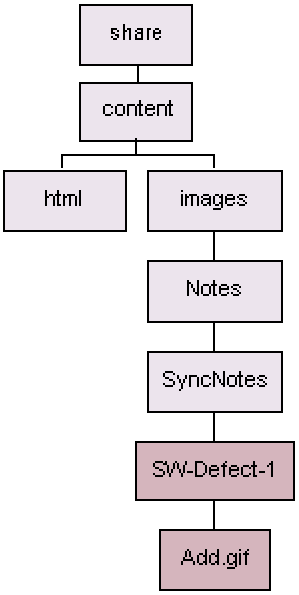
In order to programmatically customize note type panels, you work with various components on the SyncServer that make up ProjectSync note types. The SyncServer is an HTTP server process that manages shared information and performs administrative functions such as user privilege validation for DesignSync and

ProjectSync. (For more information, see DesignSync Data Manager Administrator's Guide: SyncServer Architecture.) The data of each note resides in a relational database on the SyncServer. You access this data using the ProjectSync graphical interface or commands such as `note` and `url`, described in the ENOVIA Synchronicity Command Reference. While the data for a note is stored in the relational database, most of the display information for the note is stored within the SyncServer file structure. Because users access ProjectSync on their web browsers, all of the information displayed for a note is not stored in a database, but is instead rendered as needed based on property type files, property order files, optional HTML panel layout files, and initialization scripts.

Note Type Directories: SW-Defect-1

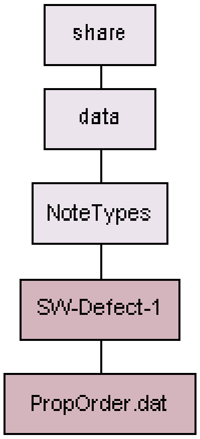
The following diagrams show the files ProjectSync uses to implement the SW-Defect-1 note type. The blocks shown in dark mauve are the files ProjectSync installs for the SW-Defect-1 note type; ProjectSync stores these files within the `<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share` hierarchy. You can make customizations to the note type files either within this custom server `share` directory or the site `share` directory (`<SYNC_SITE_CUSTOM>/share`). See (ProjectSync User's Guide: The Custom Hierarchy for more information about these directories.)

content directory



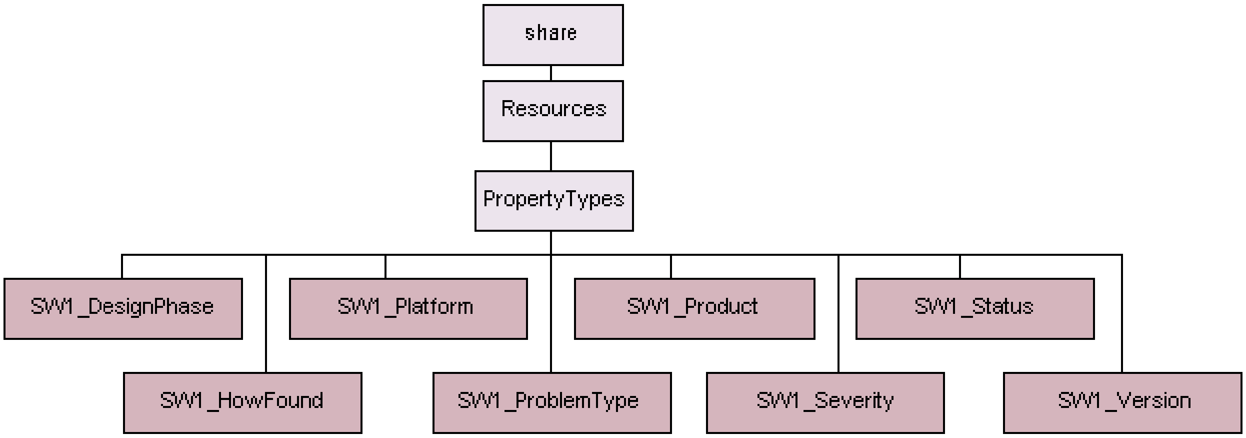
To implement the SW-Defect-1 note type, ProjectSync installs a file named `Add.gif`, an icon accompanying the SW-Defect-1 note type in the main menu. You can store your own `Add.gif` files for particular note types to customize the main menu icons. To do so, you must name the icon file, `Add.gif`, and store it in the `/share/content/images/Notes/SyncNotes/<NoteType>` directory. See ProjectSync User's Guide: Customizing the Main Menu for details.

data directory

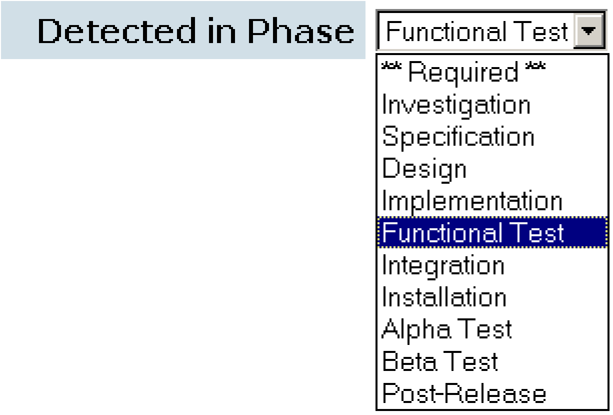


The NoteTypes directory contains a directory for each note type, within which is a `PropOrder.dat` file. The `PropOrder.dat` file lists the properties (fields) within the note type and defines the order of the properties. To change the order of fields in the note type, use the ProjectSync Note Type Manager and select **Rearrange the display order of fields on an existing note type** (ProjectSync User's Guide: Rearranging Fields on a Note Panel).

Resources directory

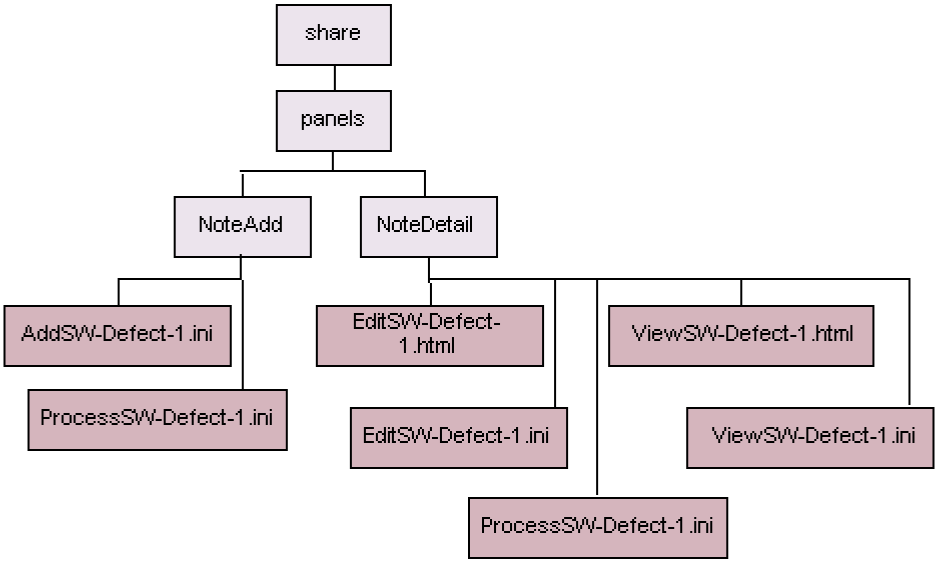


The `PropertyTypes` directory stores the definitions for custom property types. If you need new graphical elements to represent a property, such as a customized pull-down menu, you create a property type using the ProjectSync Note Type Manager and selecting **Manage property types** (ProjectSync User's Guide: Creating New Property Types). For example, the SW1_DesignPhase property type is a pull-down menu containing the phases of a software design cycle with the choices listed in a specified order:



Detected in Phase Functional Test

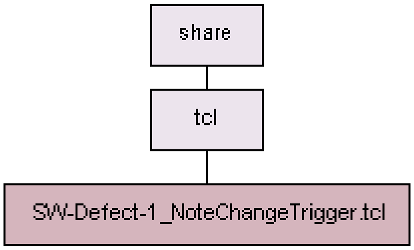
panels directory



The panels directory contains the files that implement note panels. The `NoteAdd` directory contains the files that implement the Add mode of each note panel. The `NoteDetail` directory implements the Edit and View modes of each note panel. These directories can contain an HTML panel layout file (for example, `EditSW-Defect-1.html`) and a panel initialization script (`EditSW-Defect-1.ini`) for each mode of a note type. The NoteAdd and NoteDetail directories also contain `ProcessSW-Defect-1.ini` files that customize the behavior of the panel after it is submitted. These types of panel files are described in Anatomy of ProjectSync Panels.

In addition to the SW-Defect-1 note panel files, the NoteAdd and NoteDetail directories in the `<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels` directory contain note panels for other note types that you have installed. If you customize note type panels or create custom note type panels, these panels reside in these directories, as well.

tcl directory



The `tcl` directory is the directory where you place `.tcl` files. Place your triggers or Tcl functions in the `tcl` directory. In your scripts, you can include the Tcl `locate` command to find this directory, regardless of which custom `share/tcl` directory (site or server) stores the script:

```
  source [locate share/tcl/tclUtilities.tcl]
```

When you install the SW-Defect-1 note type, ProjectSync stores the associated server trigger script in the custom server `share/tcl` directory.

Server trigger scripts control the behavior of a note type. For example, the `SW-Defect-1_NoteChangeTrigger.tcl` script sets the DateFixed field to the current date when a note enters the closed state or clears it when a note transitions from closed to open.

Anatomy of ProjectSync Panels

A ProjectSync panel is an element of the ProjectSync user interface, such as a window or a form. Panels implement the ProjectSync home page, menus, and notes. Panels generally have a 'gather' phase and a 'process' phase. The gather phase of a panel is implemented with a graphical interface that collects input from users. The process phase processes the data from the gather phase, performing data checking and displaying processing results.

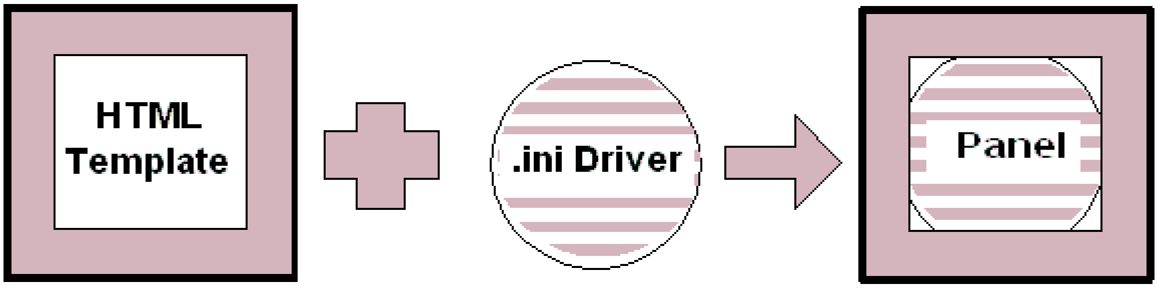
Panel Modes

A panel mode is a particular view of a panel. Panels you create can have multiple panel modes. For example, users can add, view, and edit notes. Panel modes let you select different graphical elements for different modes. For example, an Add Note panel might display a pull-down menu field with choices listed in a particular order. The View Note panel for the same note type might display a read-only text box showing the value a user has selected for the field.

Each note type mode has its own panel implementation. To customize a note type, you can customize each of these panel modes. You can

- Customize just the HTML template.
- Customize the HTML template and a corresponding panel initialization driver file.
- Replace the HTML template and initialization driver by a Tcl panel script.

Template and Driver Implementation



ProjectSync implements panels with HTML panel template (`.html`) files and optional panel initialization driver (`.ini`) files. The HTML template provides the static format of the panel and the initialization driver provides the dynamic content. You implement the fields within the HTML template by including **substitutions** for the fields, commands that insert graphical elements. You can then use any HTML commands you need to format the graphical elements of a panel within the HTML template file.

You change the behavior of a panel by editing or creating an optional panel initialization driver (`.ini`) file. The initialization driver file contains the definitions of the substitutions, the commands that generate the content for the panel. In the panel initialization driver, you can create new substitution tags for panels. Then, you embed these substitution tags within the HTML template. You can also use a panel initialization script to display different modes, or views, of a panel depending on circumstances you specify. For example, you can customize the panels for different types of users. You can also customize the behavior of a panel after a user submits the panel by creating a processing initialization driver.

Use the method of a template and initialization driver if your panel has mostly static HTML content.

Example: Creating a Custom Template and Driver

The following HTML template, `MyPanel.html`, and initialization driver, `MyPanel.ini`, generate a panel displaying the user's name and the date. Most panels you will customize will be note panels; however, for simplicity, this example generates a basic panel:

HTML Template Example: `MyPanel.html`

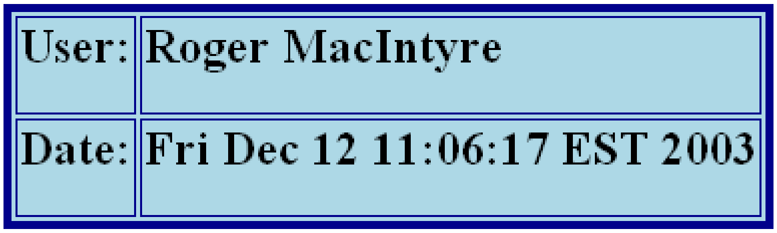
```
<html>
<head>
  <title>My Custom Panel</title>
</head>

<body>
```

```
<table border=4 bordercolor=darkblue bgcolor=lightblue>
      <tr>
  <td> <h2>User:</h2> </td>
  <td> <h2><!-- SYNC user --></h2> </td>
</tr>
<tr>
  <td> <h2>Date:</h2> </td>
  <td> <h2><!-- SYNC date --></h2> </td>
</tr>
</table>
</body>

</html>
```

Initialization Driver File: **`MyPanel.ini`**

```
substitution date {} {
  puts "[clock format [clock seconds]]"
}

substitution user {} {
  global SYNC_User
  set userUrl sync:///Users/$SYNC_User
  set name [url getprop $userUrl Name]
  puts -nonewline $name
}
```

MyPanel Output



User:	Roger MacIntyre
Date:	Fri Dec 12 11:06:17 EST 2003

The `MyPanel.ini` initialization driver file defines two custom substitutions, `user` and `date`. These substitutions are called in the HTML template using this syntax:

```
  <!-- SYNC user -->
  <!-- SYNC date -->
```

The `user` substitution uses the `$SYNC_User` global variable and the `url getprop` command to extract the user's name. The `date` substitution calls the Tcl `clock` command to obtain the date. The `MyPanel.html` file invokes these custom substitutions and also implements the look and feel of the panel.

The substitutions in `MyPanel.ini` are custom substitutions. To learn more about creating your own substitutions, see the `substitution` command. Predefined note panel substitutions and the syntax for calling them are described in Note Panel Substitution Tags.

Many ProjectSync panels are implemented using the template and initialization driver method. ProjectSync stores these files in the following directory:

```
<SYNC_DIR>/share/panels
```

Setting Up the MyPanel Sample Panel

To create the MyPanel custom panel, you store both the `MyPanel.html` and `MyPanel.ini` files in a panel directory you create specifically for the new panel in either the custom server or site hierarchy:

Server:
`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels/MyPanel`

Site: `<SYNC_CUSTOM_DIR>/site/share/panels/MyPanel`

The name of the custom panel directory must match that of the HTML template and the panel initialization file, in this case, `MyPanel`.

To load the custom panel, you enter the ProjectSync panel URL into your web browser using this syntax:

```
http://<host>:<port>/scripts/isynch.dll?panel=MyPanel
```

Replace the `<host>` and `<port>` variables above with the host and port number of your SyncServer.

See URLs for Loading Note Panels for more detailed information about loading panels, including parameters.

Note:

ProjectSync does not override the default `.ini` file with your `.ini` files in the server-specific or site-wide `panels` directory. Instead, the contents of all of the `.ini` files are merged. The more-local definitions of functions and substitutions take precedence. See Precedence of Panel Initialization Driver Files for details.

HTML Templates for Note Panel Modes

Most panels you will customize will be note panels. Like general panels, you also create HTML templates and Tcl panel initialization scripts in the custom site-wide

(<SYNC_SITE_CUSTOM>/share/panels) or server-specific (<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels) location for note panels. See ProjectSync User's Guide: The Custom Hierarchy for more information about these directories. Unlike general panels (of which the MyPanel panel is an example), note panels are stored in the NoteAdd or NoteDetail subdirectories of the panels directory.

These are the HTML templates you can include for a note type:

- NoteAdd/Add<notetype>.html - HTML template that implements the Add panel of the note type.
- NoteDetail/Edit<notetype>.html - HTML template that implements the Edit panel of the note type.
- NoteDetail/View<notetype>.html - HTML template that implements the View panel of the note type.

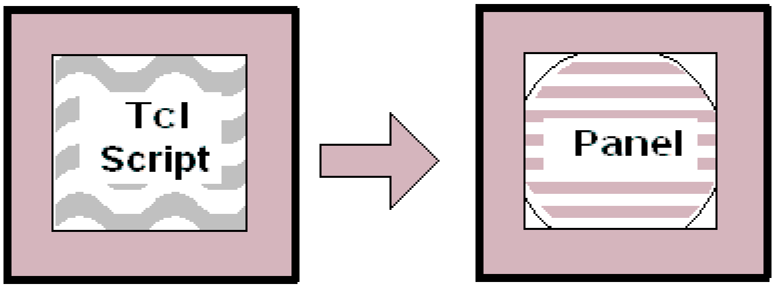
These are the initialization scripts you can include for a note type:

- NoteAdd/Add<notetype>.ini - Create this script to modify the behavior of the Add panel of the note type.
- NoteDetail/Edit<notetype>.ini - Create this script to modify the behavior of the Edit panel of the note type.
- NoteDetail/View<notetype>.ini - Create this script to modify the behavior of the View panel of the note type.

Note:

ProjectSync does not override the default .ini file with your .ini files in the server-specific or site-wide panels directory. Instead, the contents of all of the .ini files are merged. See Precedence of Panel Customization Files for details.

Straight Tcl Implementation



If your panel requires mostly dynamic HTML, you can forego the HTML template and use a Tcl script to implement the panel. In this case, you create a Tcl script that generates the entire HTML code for the panel. This method of implementing panels is for advanced Tcl programmers. Much like a web developer uses Common Gateway Interface (CGI) to implement web pages, you can use the Tcl programming language to implement ProjectSync panels. The HTML panels generated by your Tcl script can be as sophisticated as you like. You can use the Synchronicity Tcl (stcl) commands documented in the ENOVIA Synchronicity Command Reference.

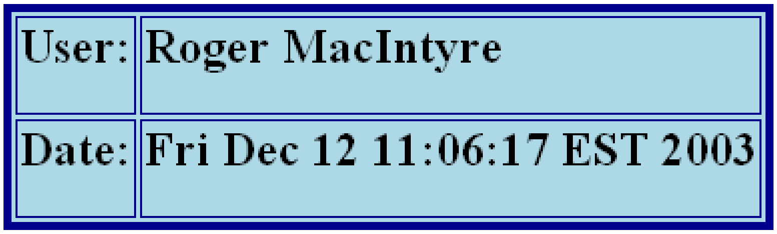
Example: Creating a Custom Tcl Panel Script

Like the `MyPanel.html` HTML template and the `MyPanel.ini` initialization driver shown above, the following Tcl script, `MyPanel.tcl` generates a panel displaying the user's name and the date.

Tcl Script Example: **`MyPanel.tcl`**

```
proc getdate {} {
  return [clock format [clock seconds]]
}

proc getuser {} {
  global SYNC_User
  set userUrl sync:///Users/$SYNC_User
  return [url getprop $userUrl Name]
}

puts "
<html>
<head>
  <title>My Custom Panel</title>
</head>
<body>
  <table border=4 bordercolor=darkblue bgcolor=lightblue>
  <tr>
    <td> <h2>User:</h2> </td>
    <td> <h2>[getuser]</h2> </td>
  </tr>
  <tr>
    <td> <h2>Date:</h2> </td>
    <td> <h2>[getdate]</h2> </td>
  </tr>
  </table>
</body>
</html>
"
```

MyPanel Output



User:	Roger MacIntyre
Date:	Fri Dec 12 11:06:17 EST 2003

You can try this example by creating the `MyPanel.tcl` file above and storing the file in the custom server or site `/share/panels` directory within a directory named `MyPanel:`

Server:
`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels/MyPanel/MyPanel.tcl`

Site: `<SYNC_CUSTOM_DIR>/site/share/panels/MyPanel/MyPanel.tcl`

You load a Tcl script panel as you load a panel implemented with a template and driver, using this syntax:

```
http://<host>:<port>/scripts/isynch.dll?panel=MyPanel
```

Replace the `<host>` and `<port>` variables above with the host and port number of your SyncServer.

To customize a panel using this method, see Implementing Panels Using Tcl Panels Scripts.

Tcl Scripts for Note Panel Modes

You create Tcl panel scripts in the same `share/panels` directories where the HTML templates for the note panels are stored. These files are located in either the site-wide (`<SYNC_SITE_CUSTOM>/share/panels`) or server-specific (`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels`) location:

- `NoteAdd/Add<notetype>.tcl` - Create this script to implement the Add panel for the note type.
- `NoteDetail/Edit<notetype>.tcl` - Create this script to implement the Edit panel for the note type.
- `NoteDetail/View<notetype>.tcl` - Create this script to implement the View panel for the note type.

Important:

If you have implemented a panel using both methods -- the template and driver method as well as the Tcl method -- the template and driver method overrides the Tcl implementation and the Tcl panel script is ignored. For more information, see Precedence of Panel Customization Files.

URLs for Loading Panels

You specify the invocation of a panel using a URL (uniform resource locator). The panel specified can be an existing ProjectSync panel, a note panel, or a custom panel. The user's web browser decodes the URL and loads the specified panel. The URL includes encoded parameters made available to the note panel. You can customize panels by setting up links or buttons that execute a panel URL, thus loading a new panel. In your Tcl scripts or initialization driver (`.ini`) files, use the `cgi_arg` Tcl utility to access the parameters passed into a note panel.

Panel URL

```
http://<host>:<port>/scripts/isynch.dll?panel=<panelmode>
&NoteType=<notetype>&DisplayMode=<displaymode>
&command=<commandmode>&<parmlist>
```

Arguments

`<host>` Specifies the machine name of the SyncServer.

`<port>` Specifies the port number. The default port number for the SyncServer is 2647.

`<panelmode>` Specifies a panel name or a note panel mode:

- `panel=NoteAdd` - The Add mode of a note panel.
- `panel=NoteDetail` - The Edit or View mode of a note panel.
- ProjectSync panel - An existing ProjectSync custom panel. For example, `panel=UserAdd` loads the User Profile panel and `panel=TextSearch` loads the Full Text Search panel.
- `panel=<custom_panel>` - One of your own panels, for example, `panel=MyPanel`.

`<commandmode>` Specifies the mode of the panel to be loaded. For note panels, the mode supported is `Process`. You can set up a process mode panel to process the results of the gather stage of a panel. For example, you can execute the processing phase of a panel by specifying `&command=Process` to load the `Process<panel>.tcl` file (or the `Process<panel>.html` and `Process<panel>.ini` files). Your processing scripts (`Process<panel>.tcl` or `Process<panel>.ini`) must follow the guidelines described in the Panel Process Phase topic.

For custom panels, you can create your own modes. You can use the `command` parameter to specify a special mode. The special mode of the panel must be stored in the following directory structure:

```
/share/panels/<custompanel>/
<custompanel><commandmode>.html
```

See Example: Loading a Special Mode of a Panel.

`<notetype>` Specifies an existing predefined or custom note type, for example, `NoteType=SW-Defect-1`.

`<displaymode>` Distinguishes between the `NoteDetail` panel modes (Optional):

- `Edit`
- `View` (default)

`<parmlist>` Specifies an optional list of parameters with name/value pairs to pass into the script: `parm1=value1&parm2=value2&parm3=value3`.

- Separate each parameter from the previous one using an ampersand (&).
- Separate the name from its value using an equal sign (=).
- The parameters and values cannot contain spaces. Replace spaces in values with the + character or enclose the entire parameter list in quotes (").
- Encode URLs using the `encodeUrl` Tcl utility.

For a list of possible parameters used with note panels, see Note Panel Arguments. You can also create your own parameters and extract their values in a `.ini` initialization driver or a Tcl script using the `cgi_arg` command.

For parameters with multiple values, repeat the parameter name for each value:
`multvalparm=value1&multvalparm=value2&multvalparm=value3`.

For example:
`&stoplights=red&stoplights=green&stoplights=yellow`.

Use the `cgi_arg -multi` option to extract a multi-valued parameter. See the `cgi_arg` command description for an example of the `-multi` option.

Example: Loading a Note Panel in Add Mode

To load a note panel, the URL you specify must include the panel's mode: `NoteAdd` for Add mode, `NoteDetail` for Edit or View mode. For example, to load the note panel of the Note note type in Add mode, you enter the following URL:

```
http://myhost:2647/scripts/isynch.dll?panel=NoteAdd&NoteType=Not
e
```

The Add Note panel displays.

Example: Loading a Note Panel in View Mode

To pass an argument to the note panel, you use the following syntax:

```
  &parm=value
```

For example,

```
http://myhost:2647/scripts/isynch.dll?panel=NoteDetail&NoteType=
Note&NoteId=1
```

Note 1 displays in View mode, the default mode for the NoteDetail panel.

Example: Loading a Note Panel in Edit Mode

If you want a note panel to display in Edit mode, you set the `panel` parameter to `NoteDetail` panel parameter (`NoteDetail`specifies Edit and View mode) and set the `DisplayMode` parameter to `Edit`:

```
http://gilmour:30048/scripts/isynch.dll?panel=NoteDetail&NoteTyp
e=
Note&DisplayMode=Edit&NoteId=1
```

Note 1 displays in Edit mode.

Example: Passing Multiple Parameters

To specify multiple parameters, you chain together `parameter=value` pairs using the & character. For example, in the following example, custom parameters `name` and `month` are specified:

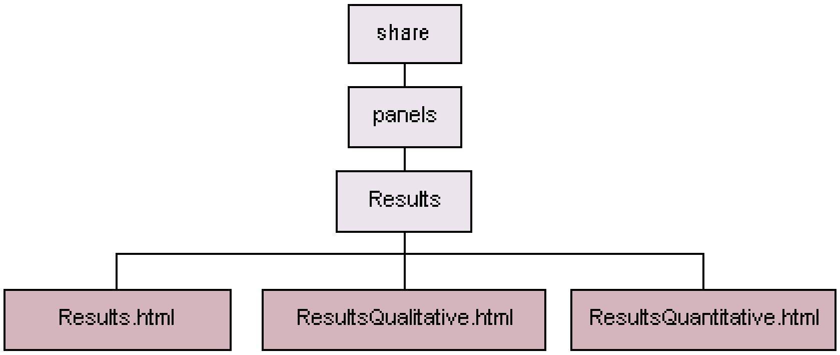
```
http://myhost:2647/scripts/isynch.dll?panel=StatusResult&name=Jo
se&month=February
```

Example: Loading a Command Mode of a Panel

For panels that are not note panels, you can use the `command` parameter to specify your own command modes. The files for the command modes must be specified as follows:

```
/share/panels/<custompanel>/<custompanel><commandmode>.html
```

The following directory structure shows a custom panel, Results, with two command modes: Qualitative (named `ResultsQualitative.html`) and Quantitative (named `ResultsQuantitative.html`).



To invoke the main Results panel, use invoke the following URL:

```
http://gilmour:30048/scripts/isynch.dll?panel=Results
```

To invoke the Qualitative command mode, you use the `command` parameter:

```
http://gilmour:30048/scripts/isynch.dll?panel=Results&command=Qu
alitative
```

Likewise, to invoke the Quantitative command mode, you use the `command` parameter:

```
http://gilmour:30048/scripts/isynch.dll?panel=Results&command=Qu
alitative
```

Precedence of Panel Customization Files

In most cases, when ProjectSync applies customizations, the more-local settings take precedence over the less-local settings. The search order is:

1. Server-specific customization files
2. Site-wide customization files
3. Enterprise-level customization files (The custom enterprise area is reserved for future development. Do not create custom files in this area.)
4. Default installation files

The following panel customization files adhere to this search order, where the more-local version of the file takes precedence:

- HTML panel template files, for example:
  `/share/panels/NoteDetail/EditSW-Defect-1.html`
- Image files, for example:
  `/share/content/images/Notes/SyncNotes/SW-Defect-1/Add.gif`
- Property (field) order files, for example:
  `/share/data/NoteTypes/SW-Defect-1/PropOrder.dat`
- Property type definition files, for example:
  `/share/Resources/PropertyTypes/SW1_DesignPhase`
- Note type configuration files, for example:
  `/share/config/YellowSticky.conf`
- Tcl panel scripts (located in the `/share/panels` directory), for example:
  `/share/panels/NoteDetail/EditSW-Defect-1.tcl`
- Tcl server-side trigger scripts, for example:
  `share/tcl/SW-Defect-1_NoteChangeTrigger.tcl`

- Other `.tcl` scripts in the `/share/tcl` directory

Precedence of Panel Initialization Driver Files

Unlike most of the customization files, ProjectSync handles the panel initialization (`.ini`) files in a cascading method. ProjectSync reads in all versions of the file and merges the contents. This is similar to how the DesignSync handles registry files. If the `.ini` files contain duplicate definitions for a function or substitution, the more-local definition takes precedence. For example, a substitution named `hi` in a panel initialization (`.ini`) file within the server hierarchy (`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels/MyPanel/MyPanel.ini`) takes precedence over the substitution of the same name in the site panel initialization file (`<SYNC_CUSTOM_DIR>/site/share/panels/MyPanel/MyPanel.ini`).

Important

If you customize a panel initialization (`.ini`) file, do not copy the entire `.ini` file from another `/share/panels` hierarchy, such as the default installation `.ini` file (for example, `<SYNC_DIR>/share/panels/NoteAdd/NoteAdd.ini`) or the server `.ini` file (for example, `<SYNC_DIR>/custom/servers/myhost/2647/share/panels/NoteAdd/AddSW-Defect-1.ini`). If you duplicate the entire `.ini` file and just modify some part of the file, the panel might execute slower and show unintended behavior. The `.ini` files can perform actions as well as store definitions; thus, duplicate `.ini` files might unintentionally repeat actions and cause unexpected behavior. Instead, only include new customizations in your custom site or server `.ini` file.

Precedence of tcl Scripts

If you have `.tcl` files with the same name in different parts of the custom hierarchy and the same setting is specified in more than one version of the file, the more-specific setting takes precedence over the wider setting. That is, settings in the site area override those in the enterprise area and settings in the server area override those in the site area. (The custom enterprise area is reserved for future development.)

Precedence of Custom Panels

ProjectSync searches the custom panel directories and the DesignSync system (defautl) directories for panel implementations:

- Server-wide customizations: `<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels/`
- Site-wide customizations: `<SYNC_CUSTOM_DIR>/site/share/panels/`
- Default panels: `<SYNC_DIR>/share/panels/`

ProjectSync loads the most local of the panel implementations. The implementation can be a single panel Tcl script (`.tcl`) or a panel template (`.html`) and initialization driver (`.ini`) pair.

Keep these points in mind when you load custom panels:

- You can implement panels using either a Tcl script or an HTML template and optional initialization driver file. (See Anatomy of ProjectSync Panels to understand these methods). If a panel has an associated HTML template as well as a Tcl panel script, the template has precedence over the Tcl panel script for implementing the panel; in this case, the Tcl panel script is ignored.
- If you have a Tcl panel script for a panel as well as an initialization driver script, you can override the initialization script's precedence; you can call the Tcl script directly. To do this, you call the `select_tcl_script` utility from within the `.ini` file:

  ```
  select_tcl_script <template_name>
  ```

  where template name is the name of the Tcl file without the .tcl extension. You can explicitly specify that the HTML template and optional initialization driver be used by calling the `select_template` command, as well.

- If initialization drivers (`.ini` files) exist for a panel in both the site and server panel directories, their contents are merged. See Precedence of Panel Initialization Driver Files for details.
- Read access is needed to the template files so ProjectSync can load them into the interpreter.

- If you make changes to panel files, you do not need to restart the SyncServer as you do with customizations such as access controls. You can make a change and then reload your web browser to verify your change. Likewise, if you are editing note panels, the next time you or a user adds, edits, or views the note, the changes display.

Precedence of Global Variables

You can set global variables in a number of locations. The following list includes the files in which you can set global variables in order of precedence:

- Note Panel Initialization Files (for example, `share/panel/NoteAdd/AddYellowSticky.ini`)

  See Coding Practices for Panel Initialization Drivers.

- Note Type Configuration Files (for example, `share/panel/config/YellowSticky.conf`)

  See Note Type Configuration Files.

- Tcl Files (for example, `share/tcl/common.tcl`)

  See Coding Practices for Tcl Panels Scripts.

Thus, if you set the `ProjectName` variable in the `AddYellowSticky.ini` file and in the `YellowSticky.conf` file, the `AddYellowSticky.ini` file takes precedence. Set the variable at the scope that makes sense for your needs. If you are setting the `ProjectName` variable in order to seed the Project field in the Add mode of a note type, it makes sense to add the variable to the initialization file for that panel mode, in this case, `AddYellowSticky.ini`. If you want to set the variable for all modes of a particular note type, set the variable in the configuration file. Unlike panel files, configuration files require a server reset if you modify them. Use the ProjectSync **Reset Server** command to reread configuration files.

Note Panel Fields

You use the ProjectSync Note Type Manager to create fields. You can then modify the fields in your HTML panel templates or your Tcl panel scripts. The type of field generated for each property depends on its property type and the panel mode -- Add, Edit, or View. The following table describes the graphical elements generated by default for the predefined property types. See ProjectSync User's Guide: Predefined Property Types for complete descriptions of these property types.

Property Type	Add Mode	Edit Mode	View Mode

Boolean	Check box	Check box	True/False in read-only text
Date The Date property type has a date component, but no time component. (The Date property type used prior to ProjectSync2.5 is now the equivalent of the Timestamp property type.)	Type-in field with calendar pop-up	Type-in field with calendar pop-up	Date in read-only text
Time The Time property type has a time component, but no date component.	Type-in field with calendar pop-up	Type-in field with calendar pop-up	Time in read-only text
Timestamp The Timestamp property type has a date and a time component. (Equivalent to the pre-2.5 Date property type.)	Type-in field with calendar pop-up	Type-in field with calendar pop-up	Date and time in read-only text
Float	Type-in field	Type-in field	Float value in read-only text
Integer	Type-in field	Type-in field	Integer value in read-only text
String#	Type-in field allowing # characters	Type-in field allowing # characters	Read-only text box
String	Type-in field allowing unlimited # characters	Type-in field allowing # characters	Read-only text box
ChoiceList	Pull-down menu	Pull-down menu	Read-only text box
StateMachine	Pull-down menu	Pull-down menu	Read-only text box
SyncClass	Pull-down menu	Pull-down menu	Read-only text box
SyncPriority	Pull-down menu	Pull-down menu	Read-only text box
SyncRevCtrlCmd	Pull-down menu	Pull-down menu	Read-only text box
SyncState	Pull-down menu	Pull-down menu	Read-only text box

SyncUserList	Pull-down menu	Pull-down menu	Read-only text box

You specify the default values of the fields for Add mode when you create the field using the Note Type Manager. If you edit the HTML template for a note panel, you can specify optional presentations of these fields. For example, you can choose to display a ChoiceList field as a pull-down menu or a series of radio button fields. To see examples of the graphical elements generated by the note type fields, see the `field` substitution description.

Configurable Fields

Configurable fields are special fields you can include in a note type:

- **cclist** – Creates a CC list field for entering the names of users to receive email notifications when the note is updated. This field includes a **Modify** button that brings up a list of all the users registered on the ProjectSync server. Users can select names from this list to insert in the CC list field. To create a CC list field, you use the property name `cclist` or `cc_list` (not case sensitive). For multiple CC list fields, specify the CC list field names in your `<notetype>.conf` file. See ProjectSync User's Guide: Adding CC List Fields to Notes for details.
- **linknotes** – Creates a field for linking to other notes. Users can enter one or more note type names and numbers to create hyperlinks to the notes. To add a linked notes field, you use the property name `linknotes`, `link_notes`, `linknote`, or `link_note`. For multiple linked notes fields, specify the link notes field names in your `<notetype>.conf` file. See Example: Links Notes Field in the `field` substitution description for an example. See ProjectSync Help: Setting Up Links Between Notes for details.

  Note: If you want to create a field that links to an object of a project other than a ProjectSync note, use an attachment substitution.

- **fileattach** – Creates a field where users can upload an attachment file to the server so that people can read or download the attached file. To add a file attachment field, you use the field name `fileattach` for a single field. For multiple file attachment fields, specify the file attachment field names in your `<notetype>.conf` file. See Example: File Attachment Field in the `field` substitution description for an example. See ProjectSync Help: Attaching Files to Notes for details.

  Note: If you are modifying an HTML panel template and your note panel contains a file attachment field, you must include the `NoteForm` substitution. The `NoteForm` substitution generates an HTML `FORM` tag with an attribute that ensures the proper operation of the file attachment fields. If you add a file attachment field when you create your note type using the ProjectSync Note Type Manager, ProjectSync adds the `NoteForm` substitution automatically. If you are implementing a custom panel that is not a ProjectSync note panel or if you want to include your own validate function within the `FORM` tag, you must include your own `FORM` tag with the `enctype` attribute shown below in bold typeface:

```
<FORM name=NoteAddForm action=isynch.dll
enctype=multipart/form-data method=post onSubmit='return
Validate(this)'>
```

- **keywords** – Creates a field for entering search keywords. This field includes a **Modify** button that brings up a list of predefined standard keywords. Use the ProjectSync Keyword Manager available from the Note Type Manager to add or edit keywords. Keywords provide another mechanism to group notes besides grouping by project, configuration, and note type. You can query on keywords to gather all the notes pertaining to a particular keyword. To add a keywords field, you use the field name `keywords` or `key_words.` For multiple keyword fields, specify the keyword field names in your `<notetype>.conf` file. See Example: Keywords Field in the `field` substitution description for an example. See ProjectSync Help: Adding Search Keyword Fields to Notes for details.
- **transcript** – Creates a type-in field where added text is appended to earlier text, rather than replacing it. To create a transcript field, you use `transcript` as your field name. For multiple transcript fields, specify the transcript field names in your `<notetype>.conf` file. See Example: Transcript Field in the `field` substitution description for an example. See also ProjectSync Help: Adding Transcript Fields to Notes for details.

The following table describes the types of graphical elements generated for configurable fields.

Configurable Field	**Add Mode**	**Edit Mode**	**View Mode**
cclist	Type-in field displaying resultant CC list members **Modify** button that brings up a list box containing all users registered on SyncServer	Type-in field displaying resultant CC list members **Modify** button that brings up a list box containing all users registered on SyncServer	Read-only list of users on CC list
linknotes	Pull-down menu containing note types	List of hypertext links to notes Pull-down menu	Read-only list of hypertext links to notes

	Type-in Id number field **Add Link** button Type-in field displaying resultant note hypertext links	containing note types Type-in Id number field **Add Link** button Type-in field displaying resultant note hypertext links	
fileattach	Type-in field displaying resultant file hypertext links **Browse** button to select file to be uploaded to SyncServer	List of hypertext links to objects of the project Type-in field displaying resultant file hypertext links **Browse** button to select file to be uploaded to SyncServer **Delete Attachment** checkbox	Read-only list of hypertext links to objects of the project
keywords	Type-in field displaying resultant keywords **Modify** button that brings up a list box containing all keywords registered for the note type or globally for the SyncServer	List of keywords Type-in field displaying resultant keywords **Modify** button that brings up a list box containing all keywords registered for the note type or globally for the SyncServer	Read-only list of keywords
transcript	Type-in field for transcript entry	Read-only text showing prior transcript entries,	Read-only text showing prior transcript entries,

	Radio buttons, **Allow HTML Tags** and **No HTML Markup Intended**	with authors and timestamps Type-in field for transcript entry Radio buttons, **Allow HTML Tags** and **No HTML Markup Intended**	with authors and timestamps

You add configurable fields to a note type in one of two ways:

- Enter the configurable property name as a new field name when you create or edit a note type using the ProjectSync Note Type Manager.
- If you are creating more than one instance of a particular configuration field, map the configuration field type to a new property name in the note type configuration file, `/share/panels/config/<notetype>.conf` (in the server or site share directory). (See Note Type Configuration Files.) Enter the new configurable property name as a new field name when you create or edit a note type using the ProjectSync Note Type Manager.

See ProjectSync Help: Configurable Property Types to learn more about setting up configurable fields.

Seeding Fields

You can seed the Project and Configuration fields for the Add mode of note panels by:

- Passing these values into a panel as parameters:

```
http://<host>:<port>/scripts/isynch.dll?panel=NoteAdd
&NoteType=Feedback&ProjectName=FeedbackSurvey&ProjectConfig
=2003
```

- Setting the `ProjectName` and `ProjectConfig` global variables:

```
set ProjectName FeedbackSurvey
set ProjectConfig 2003
```

You can set these variables in the `<notetype>.conf` file in the `/share/panels/config` directory of the custom site or server directory or in the `Add<notetype>.ini` initialization driver file:

Note Type Configuration File:

Site: `<SYNC_SITE_CUSTOM>/share/config/<notetype>.conf`

Server:
`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/config/<notetype>.conf`

See Note Type Configuration Files to learn more about this file.

Note Type Initialization File:

Site: `<SYNC_SITE_CUSTOM>/share/panels/NoteAdd/Add<notetype>.ini`

Server:
`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels/NoteAdd/Add<notetype>.ini`

If you set these variables in both the note type configuration (`.conf`) file and the note type initialization (`.ini`) file for a particular panel mode, the `.ini` file takes precedence.

Hidden Fields

ProjectSync uses HTML hidden fields to pass data to the processing phase of note panels. The `hiddens` substitution command passes the values of these hidden fields. The predefined NoteDetail and NoteAdd initialization driver files define the `hiddens` substitution for each mode such that the appropriate hidden fields are passed to the processing phase of that note panel mode. For example, the `hiddens` substitution for the Edit mode of a note panel passes the values for panel, NoteSystem, NoteType, NoteId, and the Display mode (Edit, in this case). Your HTML panel definitions should contain the `<!-- SYNC hiddens -->` substitution to ensure that these hidden field values are passed to the processing phase for each note panel mode.

You can choose to hide fields by adding the `-hidden` option to the field and std_header substitutions. See the `field` Substitution and the `std_header` Substitution.

Note Type Configuration Files

You can customize various note type settings by creating a note type configuration file. The note type configuration file is named `<NoteType>.conf` and is stored in one of the following locations:

Server:
`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/config/<NoteType>.conf`

Site: `<SYNC_CUSTOM_DIR>/site/share/config/<NoteType>.conf`

Use the note type configuration file to change the definition of a note type for all modes. For example, you can set the `ProjectName` and `ProjectC` variables to seed the Configuration field, but if you set these in the `Add<NoteType>.ini` file, the settings will only affect Add mode. Set the following types of customizations in Note Type Configuration Files:

- CC list pulldown entries
- Transcript files
- Default display language
- Attachment fields
- Link fields
- Default values for variables such as `ProjectName` and `ProjectConfig`.

See Note Panel Fields for details on setting up fields and field defaults.

Note: If you modify the note type configuration file, apply the ProjectSync **Reset Server** command so that the SyncServer reads the new configuration settings.

String Table Configuration Files

ProjectSync uses a string table to map terms used in panels to strings. The string table ProjectSync manages for English language messages is:

```
<SYNC_DIR>/share/config/strings/en.msg
```

To learn about customizing a string table file for a language besides English, please see contact a DesignSync representative.

Mapping Identifiers Using String Table Files

To map identifiers to new strings, create your own string table file in a site-wide or server-specific string table file:

Site-wide: `<SYNC_CUSTOM_DIR>/site/share/config/strings/en.msg`

Server-specific:
`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/config/strings/en.msg`

Important: Do not edit the system default string table file (`<SYNC_DIR>/share/config/strings/en.msg`).

Notes:

- You might have to create one or more of the directories in the paths above if they do not yet exist in your custom hierarchy. See ProjectSync User's Guide: The Custom Hierarchy for more information about these directories.
- If you modify the string table configuration file, apply the ProjectSync **Reset Server** command so that the SyncServer reads the new custom strings.
- The more-local `en.msg` file takes precedence over the other versions.

The following examples show some examples of using the string table to specify string values. For more examples and details, see ProjectSync User's Guide: Configuring the Wording on the GUI.

Example: Displaying a Space within a Note Type Name

Although spaces within note type names are not supported, you can have ProjectSync display a space within a note type name by mapping the name to a new string in the string table file. For example, if a custom note type named YellowSticky was created, you can create an `en.msg` string table file containing the following line:

```
NOTETYPE:YellowSticky "Yellow Sticky"
```

Store the `en.msg` file in the site-wide or server-specific `share/config/strings` directory and reset the SyncServer. The next time a ProjectSync panel displays, each instance of the note type name displays with a space as indicated in the new string. For example, after creating the `en.msg` string table file above, the **Add YellowSticky** menu item is replaced by the item, **Add Yellow Sticky**.

Example: Updating Field Names

To update field names, create a custom string table and include the following definition for the field:

```
  FIELD:<notetype>:<property>        "<new string>"
```

For example:

```
  FIELD:SyncDefect:DateFixed         "Date Completed"
```

See the default `<SYNC_DIR>/config/strings/en.msg` file for the other attributes whose strings you can modify, such as property types and choices; however, do not modify the default `en.msg` file. Instead, create a custom version in your site or servers custom area as described in section, Mapping Identifiers Using String Table Files.

Customizing Panels

Customizing Panels

ProjectSync lets you customize the user interface in varying levels of granularity. You can replace entire panels of the ProjectSync product. You can create your own panels and create links to the new panels from the main menu, the ProjectSync home page, or from other panels.

For a finer granularity of customization, you can limit your modifications to the HTML presentation of a ProjectSync panel or you can overload the substitution for a particular field in an existing panel.

Customizing Standard ProjectSync Panels

In general, the method for customizing ProjectSync panels is to place your own version of an existing panel in one of the custom `share/panels` subdirectories. If ProjectSync locates a custom panel in the custom directory hierarchy, it uses that panel instead of the default in the `$SYNC_DIR/share/panels`.

For example, you want to replace the user interface for the Add User Profiles (`UserAdd`) panel.

1. Create a `.tcl` script (or an HTML template and optional `.ini` panel initialization driver) to generate the HTML for the new panel.
2. Store your new `UserAdd.tcl` file (or `UserAdd.html` and `UserAdd.ini` files) in either of the following directories:

   Server:
   `<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels/UserAdd`

   Site: `<SYNC_CUSTOM_DIR>/site/share/panels/UserAdd`

See Precedence of Panel Customization Files for details about search order and the precedence of `.ini`, `.html`, and `.tcl` files.

To determine what you need to name the standard panel files that you wish to override, you can look in the `$SYNC_DIR/share/panels` directory to see the names of the directories that correspond to the ProjectSync panels. You can also hold your cursor over the command name in the ProjectSync main menu and view the URL that displays at the bottom of your browser window. For example, if you hold your cursor over the User Profiles:**Add** main menu button, the following URL displays at the bottom of your browser: `http://<host>:<port>/scripts/isynch.dll?UserAdd`. Thus, to replace the existing Add User Profiles panel, you can create your custom panel files in a custom directory named share/panels/UserAdd.

Creating New Panels

You can create your own panels from scratch, as well, and invoke these panels from the main menu, the ProjectSync home page, or from any other panel. In this case, you create a new directory in a site or server share/panels directory named for the panel. For example, if your new panel is to be named ListBugReports, you create one of the following directories:

Server:
<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels/ListBugReports

Site: <SYNC_CUSTOM_DIR>/site/share/panels/ListBugReports

Within the ListBugReports directory, create either a Tcl script or an HTML template and optional .ini initialization driver pair to implement the panel. These files must be named ListBugReports.tcl, ListBugReports.html, and ListBugReports.ini respectively. For an example, see Anatomy of ProjectSync Panels.

Securing Custom Panels

DesignSync panels that perform data modifications (adding information, removing information, changing information, server or access control resets) are secured against external attacks that could compromise data by including a security token that is required in order for the data modification to be accepted. Custom forms that perform data modifications can be secured in the same way.

Note: If you choose not to secure the custom panels, or if you have a staged rollout of existing custom panels, the existing panels continue to work even without adding the security token handshake.

There are two types of panels that modify data:

- Dialog (or Form) Panels - which require the user to enter some information and submit the information to the server in order to affect change. For example, creating a user, requires you enter information and click a button to submit.
- Action Buttons - in which there is no user-submitted information, the act of submitting the change is the only thing required. For example, if a server reset uses an action button to launch the reset action.

Securing Dialog Panels

DesignSync provides a procedure that sets the security token field for dialog (or form) panels.

Using the DesignSync provided Procedure:

1. In order to take advantage of the DesignSync provided procedure, add this procedure call to your code:

   ```
   proc htmlFormBegin
   ```
2. To validate the token after the data is submitted, add this call to the top of the processing .tcl code.. If there is an iterative process, for example if you collect data on multiple pages, this code should be inserted into the final process; the one that commits the changes to the server.

   ```
   validateDialogToken
   ```

Adding the Security Token without using the DesignSync provided procedure:

You can also manually insert the security token into your panel code. In the HTML file, within the <FORM...> section, most DesignSync panels, and templates have a <!-- SYNC hiddens --> section. The security token call, syncDialogTokenField is placed within the TCL code or INI that adds hidden fields to the form.

Here are a few possible formats for including the token:

Format Option 1:

```
substitution hiddens {} {

    puts "<INPUT type=hidden name=panel value=PanelName>"

    puts "<INPUT type=hidden name=mode value=Process>"

    puts [syncDialogTokenField]

}
```

Format Option 2

```
substitution hiddens {} {

    puts [htmlHiddenField panel PanelName]

    puts [htmlHiddenField command Process]

    puts [syncDialogTokenField]
```

```
}
```

Format Option 3

```
substitution hiddens {} {

    hidden panel EmailMgrAdmin

    syncDialogTokenField 1

}
```

To validate the token after the data is submitted, add the following call to the top of the processing .tcl code..If there is an iterative process, for example if you collect data on multiple pages, this code should be inserted into the final process; the one that commits the changes to the server.

```
validateDialogToken
```

Securing Actions

Some panels have a button that performs an action that performs data modification, such as the server reset commands. Since there isn't a form to act on, the security token should be included in the URL submitted by the action button, for example, your TCL code should look something like this, where the link is set:

```
set link $serverUrl?panel=PanelName&command=Action
```

And the security token is appended to the link:

```
append link & SYNC_Dialog_Token = [_httpsession getdialogtoken]
```

Note: If there are no arguments to the URL, as shown above, append the link with ? instead of &, for example:

```
set link $serverUrl?panel=PanelName&SYNC_Dialog_Token =
[_httpsession getdialogtoken]
```

As with the Secure Dialog Panels, add this call to the top of the processing .tcl code to validate the token.

```
validateDialogToken
```

Related Topics

Customizing Panels

Panel Gather Phase

The **gather** phase of panel processing is the phase where the user's web browser displays the HTML layout of the note panel mode: the Add, View, or Edit panel for the note type. The substitutions defined in an initialization driver (for example, `AddSyncDefect.ini`) generate the dynamic behavior of graphical elements. The user makes selections and enters information in the panel. The user can perform various actions such as entering text, selecting values from pull-down menus, and selecting dates from calendar pop-up fields.

The gather phase of a panel is implemented with customization files named `<panel>.tcl` (or `<panel>.html` and `<panel>.ini`).

During the gather phase, ProjectSync

- Validates access controls to ensure that the user can access the panel.

- Loads any custom HTML templates, panel initialization drivers (`.ini` files), or Tcl files, and performs error checks. See Precedence of Panel Customization Files to understand the order in which ProjectSync loads these files.
- Generates substitutions within the HTML templates.
- Loads attachments. For example, ProjectSync loads a project attachment by assembling the list of projects on the SyncServer in order to generate the choice list of the Project pull-down menu.
- Displays the panel in the user's browser.
- Displays the input the user enters and the settings the user applies to graphical elements such as pull-down menus and check boxes.

Customizations for the Gather Phase

You can customize the gather phase of a note panel in these ways:

- Add, edit, or remove fields of a note panel.

  You add, edit, and remove fields of a note panel using the ProjectSync Note Type Manager. See ProjectSync Help: Editing Note Types to learn how.

  Note: If you have generated HTML templates to customize a note panel, create new fields using the ProjectSync Note Type Manager, then add the new fields to the HTML templates by hand. To learn about the ProjectSync commands (substitutions) you use to add or modify fields, see field Substitution.

- Change the string displayed for a note type, field, property, or choice.

To change the strings displayed, you create a custom string table configuration file. See String Table Configuration Files for details. Use the string table configuration file to change all types of strings in panels. You can also change prompt strings by adding a new prompt string to the panel initialization (`.ini`) file. See Creating Panel Initialization Drivers for an example.

- Change the order of fields in a note panel.

  To rearrange fields, use the Note Type Manager. See ProjectSync User's Guide: Rearranging Fields on a Note Panel. If you want to rearrange the fields after you have completed your note type design, you can instead call the `setFieldOrder` utility from a `.ini` panel initialization file or a Tcl script.

- Change the appearance of a note panel by editing its HTML.

  You can generate HTML templates for your note types and then use HTML tags to modify the layout of the note panel. See Customizing HTML for Panels. You can also add substitution commands to your HTML template. See What Are Substitution Tags? to learn about substitutions.

- Change the property type definition underlying a field.

  The property type dictates the type of graphical element a field substitution generates. To learn about note property types, see ProjectSync User's Guide: Predefined Property Types.

- Add graphical icons to the ProjectSync menu.

  You can create your own icons for the main menu and link these to existing note panels or to your own Tcl scripts. See ProjectSync Help: Customizing the Main Menu.

- Change the substitutions called from the panel's HTML file.

  For the definitions of the substitutions you can include in note panels, see Note Panel Substitution Tags.

- Add new substitutions to the panel's initialization driver file.

  To create new substitutions, the commands that generate panel content, see the substitution command.

- Add custom documentation for your custom note types.

You can install your own documentation for each custom note type so that the **Help** button of an Add Note panel links to your custom documentation. See ProjectSync User's Guide: Adding Documentation for Custom Note Types.

Panel Process Phase

The **process** phase for note panels is the phase when ProjectSync processes the user's input. The process phase occurs after the user submits the panel. ProjectSync generates a URL string that loads the process phase of the panel. For an Add note panel, the process phase performs error checking and adds the note to the SyncServer note system. For an Edit note panel, the process phase makes the user-specified changes to the note.

During the process phase, ProjectSync

- Validates access controls to ensure that the user has the necessary permissions to perform the requested action on the note.

- Loads the custom process driver if one exists.

  To create a custom process driver, store your substitutions and Tcl procs in a file in the site or server custom area named `/share/panels/NoteAdd/Process<NoteType>.ini` (to process a note after it is created) or `/share/panels/NoteDetail/Process<NoteType>.ini` (to process a note after it is edited).

- Performs preprocessing checks.

  ProjectSync carries out preprocessing checks before loading any information about the note being processed. Store your own preprocessing code in a proc named `Process_<NoteType>_PreSchema` and store it in a file in the site or server custom area named `/share/panels/NoteAdd/Process<NoteType>.ini` file or `/share/panels/NoteDetail/Process<NoteType>.ini`. The preprocessing checks allow the user to cancel out of note processing immediately if necessary, so do not include extensive data processing in your preprocessing code. During the preprocessing phase, the data available for checking is limited to the `SYNC_Parm` and `SYNC_ClientInfo` global variables. If your checks detect errors, ProjectSync displays a failure panel.

- Fetches the note schema and loads attachments.
- Performs postprocessing checks.

ProjectSync carries out postprocessing checks after loading the note schema but before creating or updating the note. Store your own postprocessing code in a proc named `Process_<NoteType>_PostSchema` and store it in a file in the site or server custom area named `/share/panels/NoteAdd/Process<NoteType>.ini` file or `/share/panels/NoteDetail/Process<NoteType>.ini`. During the postprocessing phase, you can perform extensive checks on the data supplied in the note panel form by the user. If your checks detect errors, ProjectSync displays a failure panel.

- Creates or modifies the note.
- Processes note attachments.
- Displays the resulting note panel in the user's browser.

  By default, a ProjectSync success panel displays. You can choose to redirect processing to a specified HTML template.

Customizations for the Process Phase

You can customize the process phase of a note panel to:

- Perform data checking in order to efficiently cancel an operation such as adding or editing a note if necessary.
- Display a different panel based on criteria you specify.
- Generate informative diagnostic or success panels depending upon the results of processing.

For an example that shows how to use the process phase to modify notes, see Postprocessing in Panel Initialization Files.

Customizing Panels: Step by Step

If you decide to customize a panel by modifying its HTML template or initialization driver, follow the steps described below. For more advanced applications, you might want to override the HTML panel template completely by developing a Tcl script that performs the desired behavior; see Coding Practices for Tcl Panels Scripts for guidelines.

The basic steps for customizing ProjectSync note panels are:

1. Generate HTML templates.

   ProjectSync lets you generate the HTML templates for any installed note type, including custom note types you have created. Use the ProjectSync Note Type

Manager to generate the HTML templates. You can choose to generate the templates in the site or server custom area. See Generating HTML Templates.

2. Customize the HTML templates.

   After you generate the HTML templates, you can edit them in a text editor or a WYSIWYG HTML editor. You customize the look and feel of the note panel by modifying its HTML representation. See Customizing HTML for Panels for details. Note that you do not have to add fields manually within the HTML templates; if you want to add a field to a note panel, you can update its note type directly using the Note Type Manager before generating the HTML templates (ProjectSync User's Guide: Editing Note Types).

3. Create or modify the Tcl panel initialization driver.

   Change the behavior of a note panel by creating a Tcl panel initialization script. Each panel has a `.ini` panel initialization driver that contains the commands, or **substitutions**, reflected in the panel's HTML representation. See Coding Practices for Panel Initialization Drivers for details.

Note: The appearance of the SW-Defect-1 and BugReport note types cannot be customized in this way because these note type already use the template mechanism to implement the behavior of different views for some properties.

HTML Panel Templates

Generating HTML Templates

The Note Type Template Generator panel lets you generate the HTML templates for a note type. After you have generated the templates, you can customize them to modify the layout of the note type panels. If you generate HTML templates in the site-wide or server-specific `panels` directory, ProjectSync adheres to these custom HTML files to display the note panels instead of those specified in the default ProjectSync `panels` directory.

Use the Note Type Template Generator for extensive customizations of note types. To simply rearrange fields, use the Note Type Manager. See ProjectSync User's Guide: Rearranging Fields on a Note Panel.

Important: After you have generated HTML templates for a note type, you cannot use the Note Type Manager to rearrange the fields for the custom note type. Also if you use the Note Type Manager to add or remove a field, you must add or remove the field in each of the corresponding panel mode HTML templates (the Add, View, and Edit mode of the panel).

Warning: Do not attempt to generate templates for note types derived from the built-in note types SW-Defect-1 or BugReport, as errors will result.

To generate the HTML templates for a note type:

1. Select **NoteType Manager** from the ProjectSync menu.
2. From the list of related tasks, select **Generate HTML note type templates for a note type**.

   The Note Type Template Generator panel appears.

3. From the **Note Types** list, select the note type you want to customize.

   The **Note Types** list also displays the existing custom templates if you have already generated HTML templates for a note type.

4. Select the **Location** for the note type's HTML templates.

   If you want your template customizations to be available across all the servers on your site, select **Site**. If you want your customizations to be available only to users on the current server, select **Servers area**.

5. Keep the **Backup** button checked to ensure that ProjectSync backs up any existing templates for the note type.

6. Select the **Create Template** button to generate the HTML templates.

   ProjectSync generates three templates in the `share/panels` directory of either the site-wide (`<SYNC_SITE_CUSTOM>/share/panels`) or server-specific (`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels`) location you specified:

   - `NoteAdd/Add<note_type>.html` - Edit this file to modify the Add mode of the note type.
   - `NoteDetail/Edit<note_type>.html` - Customize this panel to modify the Edit mode of the note type.
   - `NoteDetail/View<note_type>.html` - Customize this panel to modify the View mode of the note type.

Note: After you have generated an HTML template for a note type, you cannot use the Note Type Manager's Order NoteType Properties panel to rearrange the fields for the custom note type.

Customizing HTML for Panels

You can customize the appearance of a panel by editing its HTML template using either a web page (WYSIWYG) editor or a text editor. To customize a panel, you can create, move, edit, or delete fields and their associated ProjectSync substitution tags. See What Are Substitution Tags? for a description of these tags.

This topic describes how to customize the appearance but not the behavior of fields in panels. To learn how to customize the behavior of fields, see Creating Panel Initialization Drivers.

Editing Panel HTML Templates

Before editing the HTML templates for your note panels, you must generate the templates by following the steps in the topic, Generating HTML Templates.

ProjectSync generates the HTML templates for you to edit in the following locations depending on whether you opted to generate the HTML templates in a site or server directory:

- Server-specific location: `<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels/<panel>/<panel>.html`
- Site-wide location: `<SYNC_SITE_CUSTOM>/share/panels/<panel>/<panel>.html`

After you have generated HTML templates in the site-wide or server-specific `panels` directory, ProjectSync adheres to these custom HTML files to display the note panels instead of those specified in the default ProjectSync `$SYNC_DIR/share/panels` directory.

Important: If you later add fields to the note type definition as described in ProjectSync User's Guide: Editing Note Types, you must manually add the fields to your custom templates for the note type. ProjectSync does not automatically add the fields to the HTML templates in the `custom` area.

You can edit your HTML templates using:

- A web page (WYSIWYG) editor
- A text editor

An example of editing a note panel is illustrated in Example: Customizing a New Note Type.

Editing HTML Templates Using a WYSIWYG Editor

When you load an HTML note panel template into a WYSIWYG editor, you see the basic layout of the panel. The fields display in a table. You can use the WYSIWYG editor to move, modify, delete, and create fields.

Here are some general guidelines for editing the template with a WYSIWYG editor:

- Be sure to turn on the viewing of special tags in your editor. It is very important that you do not inadvertently delete the existing substitution tags in your HTML templates.
- There are particular substitution tags that are required properties on all note types. Do not delete these substitution tags. See ProjectSync User's Guide: Required Properties for a list of these properties.
- WYSIWYG editors often break up long lines with newlines. If your WYSIWYG editor inserts newlines within comments, the substitution tags will be corrupted and the panel will not display or behave correctly in ProjectSync.
- An alternate representation for substitution tags is supported by ProjectSync. Substitution tags can be implemented with this syntax: `%(Title)%` instead of the HTML comment syntax: `<!-- SYNC field Title -->`. We recommend that you do not use the alternate syntax, `%(X)%`, if you are editing with a WYSIWYG editor because most WYSIWYG editors freely insert newlines into text strings. A newline within the substitution tag corrupts it and prevents the panel from displaying properly.

Editing HTML Templates Using a Text Editor

You can also edit the HTML templates using a text editor if you have a working knowledge of HTML. We recommend that you use an editor that highlights HTML syntax elements, such as emacs.

Edit the HTML template files as you would edit other HTML documents, keeping in mind these guidelines:

- Substitution tags are unlike standard HTML code, in that white space and newlines are not automatically removed. The following is an example of a substitution tag:

  ```
  <!-- SYNC field Title -->
  ```

  There must be a single space before and after the SYNC keyword in the substitution tag. There can be no newlines in substitution tags.

- Substitution tags can also use this syntax:

  ```
  %(field Title)%
  ```

  As in the HTML comment syntax, there can be no newlines in these substitution tags.

- ProjectSync uses tables to display lists of fields. The header cell of a row contains a prompt and the data cell contains the substitution tag. Below is an example of a table row:

  ```
  <TABLE>
    <TR>
      <TH>Responsible</TH>
      <TD><!-- SYNC field Resp --></TD>
    </TR>
  </TABLE>
  ```

Example: Customizing a New Note Type

To modify an HTML template:

1. Create your own custom note type using the Note Type Manager by selecting **Create a new note type from scratch**.
2. After you create the note type, generate HTML templates for it by following the steps in the topic, Generating HTML Templates. You can generate the templates in the site-wide or server-specify area.
3. Edit the template files for the Add, View, or Edit mode of the note, for example:

   ```
   share/panels/NoteAdd/AddYellowSticky.html
   ```

```
share/panels/NoteDetail/EditYellowSticky.html

share/panels/NoteDetail/ViewYellowSticky.html
```

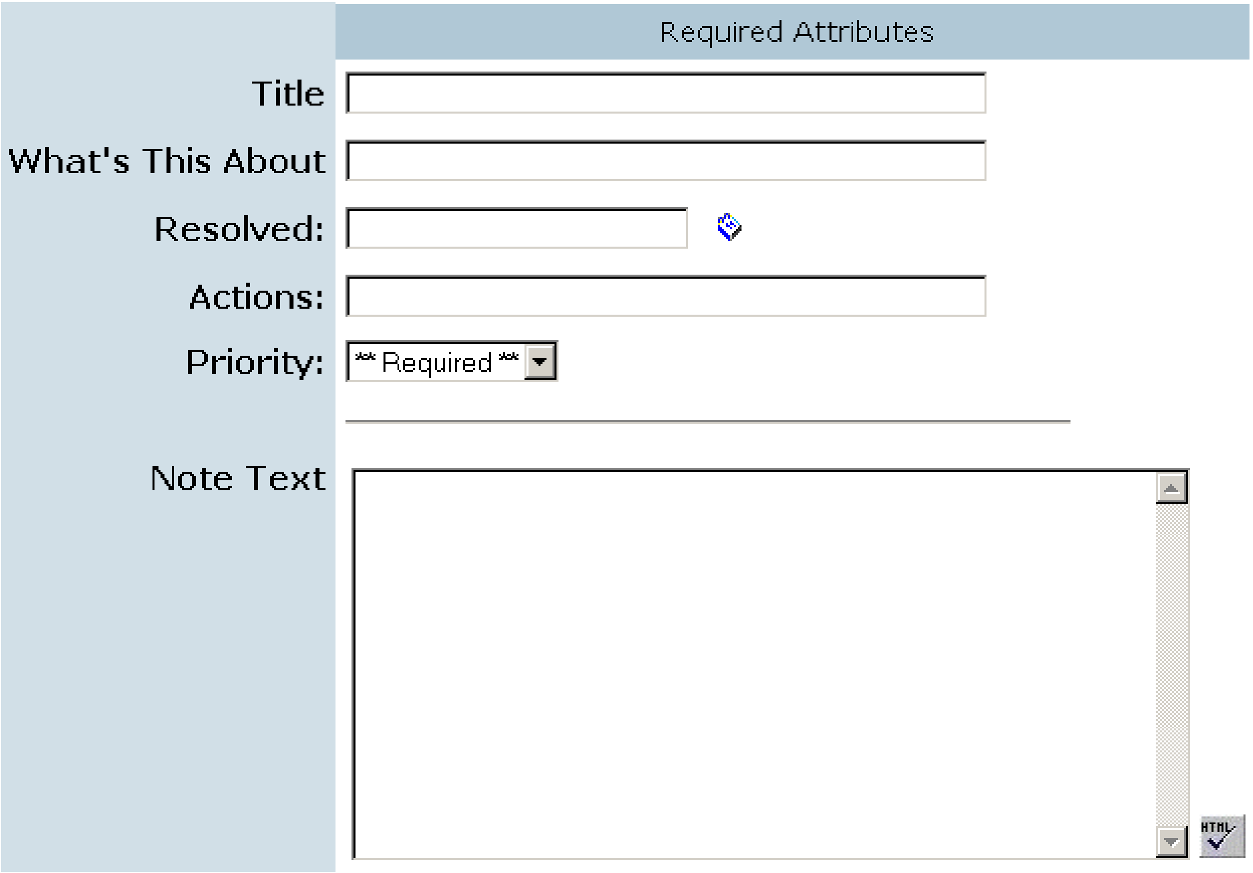
Generally, if you make a change to one of these panel modes, you need to make a corresponding change to the other panel modes.

4. Try rearranging fields or adding a new substitution within the templates by editing the HTML.

   To learn about the predefined substitutions available, see Note Panel Substitution Commands.

Example: Adding a Column to a Layout

The following example shows a custom note type before and after the HTML was modified. Following are the required attributes of a sample custom note type, YellowSticky, in Add mode:



Here is the HTML template code segment from the custom site or server `/share/panels/NoteAdd/AddYellowSticky.html` template for the fields shown above:

```
...
<!-- SYNC std_separator -tdclass SEPARATOR -mapped
REQUIRED_ATTRS -->

<TR>
  <!-- SYNC std_header -prompt Title -->
  <TD>
    <!-- SYNC field Title -->
  </TD>
</TR>

<TR>
  <TH align=right  bgcolor=#CCDDD0><!-- SYNC prompt Subject --
></TH>
  <TD>
    <!-- SYNC field Subject -->
  </TD>
</TR>

<TR>
  <TH align=right  bgcolor=#CCDDD0><!-- SYNC prompt Resolved --
></TH>
  <TD>
    <!-- SYNC field Resolved -->
  </TD>
</TR>

<TR>
  <TH align=right  bgcolor=#CCDDD0><!-- SYNC prompt Actions --
></TH>
  <TD>
    <!-- SYNC field Actions -->
  </TD>
</TR>

<TR>
  <TH align=right  bgcolor=#CCDDD0><!-- SYNC prompt Priority --
></TH>
  <TD>
    <!-- SYNC field Priority -->
  </TD>
</TR>

<TR>
  <!-- SYNC std_header -->
  <TD><HR width=80%></TD>
</TR>
```

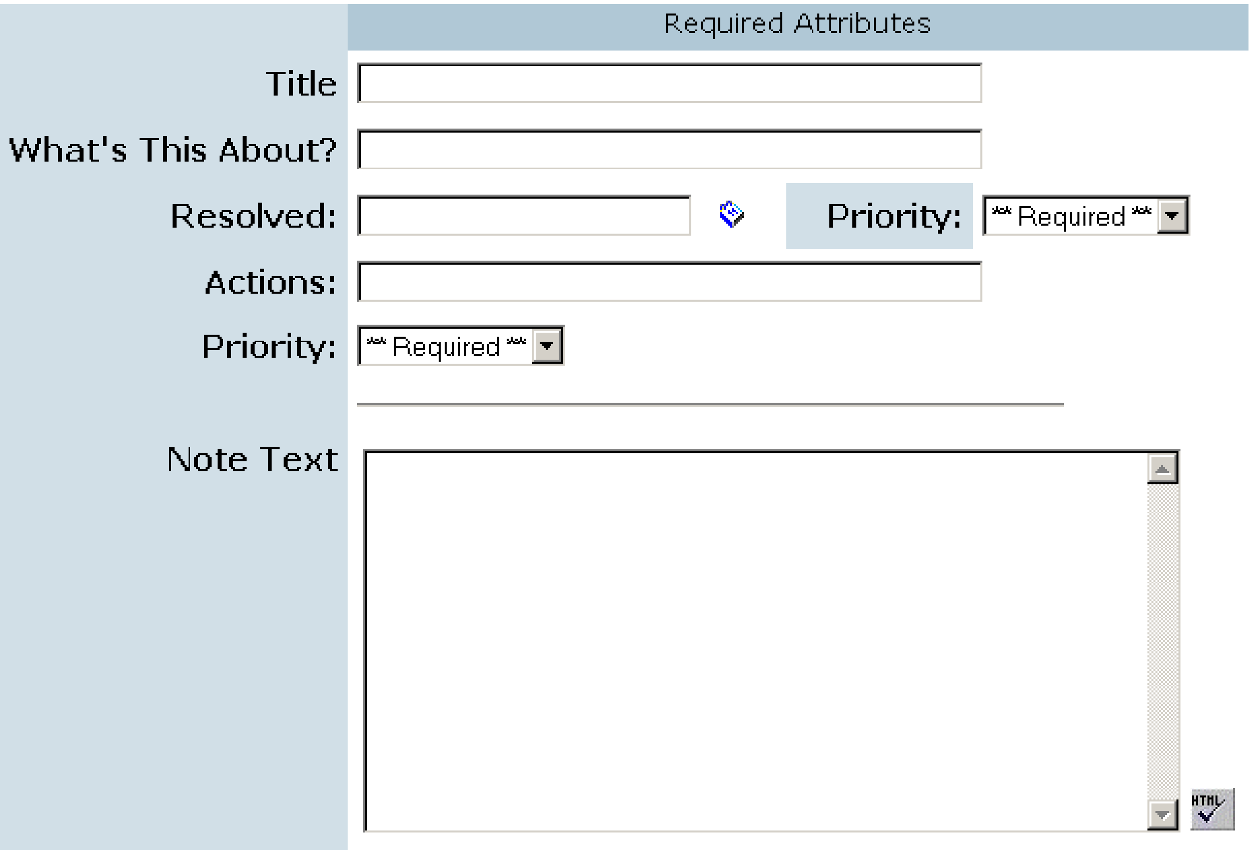
```
<TR>
  <!-- SYNC std_header -prompt Body -valign top -->
  <TD>
    <!-- SYNC field Body -rows 12 -->
  </TD>
</TR>
```

We modify the above note type so that the small fields, Resolved and Priority, lie side by side by including both fields in a single row. The new row is shown in bold below. The remaining rows require a column span of 3 to align the new column properly (also shown in bold).

```
<!-- SYNC std_separator -tdclass SEPARATOR -mapped
REQUIRED_ATTRS -colspan 3 -->

<TR>
<!-- SYNC std_header -prompt Title -->
  <TD COLSPAN=3>
    <!-- SYNC field Title -->
  </TD>
</TR>

<TR>
<TH align=right  bgcolor=#CCDDD0><!-- SYNC prompt Subject --
></TH>
  <TD COLSPAN=3>
    <!-- SYNC field Subject -->
  </TD>
</TR>

<TR>
<TH align=right  bgcolor=#CCDDD0><!-- SYNC prompt Resolved --
></TH>
<TD>
<!-- SYNC field Resolved -->
</TD>
<TH align=right  bgcolor=#CCDDD0><!-- SYNC prompt Priority --
></TH>
<TD>
<!-- SYNC field Priority -->
</TD>
</TR>

<TR>
<TH align=right  bgcolor=#CCDDD0><!-- SYNC prompt Actions --
></TH>
<TD COLSPAN=3>
```

```
<!-- SYNC field Actions -->
</TD>
</TR>

<TH align=right  bgcolor=#CCDDD0><!-- SYNC prompt Priority --
></TH>
<TD COLSPAN=3>
<!-- SYNC field Priority -->
</TD>

<TR>
<!-- SYNC std_header -->
<TD COLSPAN=3><HR width=80%></TD>
</TR>

<TR>
<!-- SYNC std_header -prompt Body -valign top -->
<TD COLSPAN=3>
<!-- SYNC field Body -rows 12 -->
</TD>
</TR>
```

The panel now appears as follows:



Required Attributes

Title

What's This About?

Resolved:

Priority: ** Required **

Actions:

Priority: ** Required **

Note Text

Panel Initialization Drivers

Creating Panel Initialization Drivers

Panel initialization driver (.ini) files contain the Tcl code that drive the panel templates. You create Tcl panel initialization scripts in the `share/panels` directories where the HTML templates for the note panels are stored. These templates and initialization scripts are located in either the site-wide (`<SYNC_SITE_CUSTOM>/share/panels`) or server-specific (`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels`) location. These are the initialization scripts you can include for a note type:

- `NoteAdd/Add<note_type>.ini` - Create this script to modify the behavior of the Add mode of the note type.
- `NoteDetail/Edit<note_type>.ini` - Create this script to modify the behavior of the Edit mode of the note type.
- `NoteDetail/View<note_type>.ini` - Create this script to modify the behavior of the View mode of the note type.

You use Tcl panel initialization scripts to:

- Create new substitutions
- Select between different HTML panel templates

See Special Considerations for more general guidelines for using Tcl panel initialization scripts.

For examples of panel initialization scripts and custom substitutions within them, see the examples in the following topics:

- Anatomy of ProjectSync Panels
- AllProjectsList
- AllProjectsMap
- AllUsersList
- AllUsersMap
- Design Tool Process Survey
- substitution

Changing the Prompt String

You can override the default prompt strings for a note type in its `.ini` files. To specify your own prompt strings, add the following definition to the `.ini` files for the note type:

```
set prompts(<property_name>) <prompt_string>
```

For example, suppose that you want to rename the **Note Text** prompt in the default SyncDefect note type. This prompt is set by the Body property. To rename **Note Text** to **Comments**, add the following line to the `.ini` files for the SyncDefect note type:

```
set prompts(Body) Comments
```

You can also change a prompt string by creating a string table configuration file. See String Table Configuration Files.

Creating New Substitutions

You can create new substitution tags with new behaviors that are different from those of the standard HTML substitution tags described in Customizing HTML for Panels.

You create new substitution tags by calling the `substitution` utility. If a substitution definition for a new tag is included in a note panel's `.ini` file, you can then include that substitution tag in the corresponding HTML template file. See the `substitution` utility for examples.

Selecting Between HTML Templates

You can include any Tcl code in a panel initialization script, not just substitution definitions. A common use for the Tcl initialization script is to choose between HTML templates that implement the same note panel. For example, some of your users might see one template and others might see a different template depending on criteria you define. You might also use access control information as the criteria by which ProjectSync determines which HTML template to display. See the `select_template` utility for an example.

For example, for the EditNote template, you might specify the following line in your `EditNote.ini` file:

```
  select_template UserEditNote
```

where `UserEditNote.html` is the template that implements the Edit Note panel. If a `UserEditNote.ini` file exists, ProjectSync uses this file as the panel initialization driver.

Special Considerations

- ProjectSync uses standard `.ini` files of its own for note types and stores them in the `<SYNC_DIR>/share/panels` directory. ProjectSync does not override the default `.ini` file with your `.ini` files in the server-specific or site-wide `panels` directory. Instead, the contents of all of the `.ini` files are merged.

- Do not use `puts` statements at the top level of a `.ini` file. In a `.ini` file, you can use `puts` statements only inside `substitution` subroutines or in procs called from within substitution subroutines.
- You can have ProjectSync abort template processing, for example, if the current user does not have permission for an operation. In this case, you can include a `puts` statement at the top level of your Tcl script, as long as the `puts` statement is followed by the `quit` command to prevent the HTML template from being substituted and generated.
- There are two types of initialization drivers:

  Gather phase initialization driver files. See Panel Gather Phase.

  Process phase initialization driver files. See Panel Process Stage.

Postprocessing in Panel Initialization Files

You can create your own procedures to process a note after the user submits it. To do so, create a Process initialization file in the site or server custom area.

- To process a note before it is created, create the following file:

  `/share/panels/NoteAdd/Process<NoteType>.ini`

- To process a note before it is modified, create the following file:

  `/share/panels/NoteDetail/Process<NoteType>.ini`

Note: If you want to add a procedure to process the predefined SyncDefect, HW-Defect-1, or the SW-Defect-1 note types, you must add your procedure to the existing Process initialization files that ProjectSync stores upon their installation in the custom `servers/<host>/port/share/panels/NoteAdd` and `servers/<host>/port/share/panels/NoteDetail` directories. The files are named `ProcessSyncDefect-1.ini`, `ProcessHW-Defect-1.ini`, and `ProcessSW-Defect-1.ini`, respectively.

PreSchema Processing Procedures

You can create a procedure that processes the note immediately upon note submission, before ProjectSync processes the note and accesses the notes database. This type of procedure is a PreSchema procedure. You use PreSchema procedures to perform error checking so that you can abandon note processing immediately if an error is found in the user's input.

You include a PreSchema procedure in your `Process<NoteType>.ini` file naming the procedure `Process_<NoteType>_PreSchema`. The following example shows a PreSchema procedure for the TimeCard notetype, stored in the following file:

```
  /share/panels/NoteDetail/ProcessTimeCard.ini
```

PreSchema procedures have access to only the following global variables:

- `SYNC_Parm`
- `SYNC_ClientInfo`

Example: PreSchema Procedure

The following example checks the number of hours a user has entered in the Hours field and provides an error message, abandoning note processing if the number of hours is greater than 40.

```
proc Process_TimeCard_PreSchema {} {
  global SYNC_Parm

  set hoursworked SYNC_Parm(Hours)
  if {$hoursworked > 40} {
  error "Working more than 40 hours is not allowed at this
time."
  }
}
```

PostSchema Processing Procedures

You can create a procedure to process a note after the user submits it and after ProjectSync loads the note schema, but before ProjectSync adds or modifies the note in the notes database. This type of procedure is a PostSchema procedure. Unlike PreSchema procedures, PostSchema procedures can access the note schema with the full range of global variables.

You include a PostSchema procedure in your `Process<NoteType>.ini` file naming it `Process_<NoteType>_PostSchema`. The following example shows a PostSchema procedure for the Note note type, stored in the following file:

```
  /share/panels/NoteDetail/ProcessNote.ini
```

Example 1: PostSchema Procedure adding current user and date

Upon submission of a Note, the following example fills in the LastModBy and LastModDt fields with the current user and the current date.

Note: To try out this postprocessing procedure, install and modify the predefined Note note type, adding two fields: LastModBy (of type SyncUserList) and LastModDt (of type Date). Store the following procedure in file `/share/panels/NoteDetail/ProcessNote.ini` in your server or site area, and reset the server. To test out the procedure, add and then edit a Note.

The `Process_Note_PostSchema` procedure obtains the current timestamp by accessing the `getCurrentApp` utility in the `::sdate` namespace. You do not have to source the `sdate` package, `$SYNC_DIR/share/tcl/sdate/sdate.tcl` directly; ProjectSync sources the package at start-up and exports the date utilities. You can also specify the `-pclass time` option with `getCurrentApp` to return just the time or use the `-pclass date` option with `getCurrentApp` to return just the date.

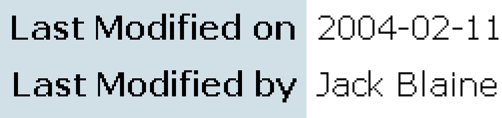
```
proc Process_Note_PostSchema {} {
  global field_values
  global SYNC_Parm
  global SYNC_User

#### Calculate the date for PS use
  set SyncDate [::sdate::getCurrentApp]

#### Make sure that the fields to store LastMod
#### info are available
  set fieldNames [note schema -ptypes noteTypes Note]
  if {[lsearch -exact $fieldNames LastModBy] == -1} {
    error "Could not find field LastModBy"
  }

  if {[lsearch -exact $fieldNames LastModDt] == -1} {
    error "Could not find field LastModDt"
  }

#### Set the values
  set SYNC_Parm(LastModDt) $SyncDate
  set SYNC_Parm(LastModBy) $SYNC_User
}
```

To complete the example, you convert the LastModBy and LastModDt fields into read-only fields by creating a custom HTML panel. To do so, use the Note Type Manager to generate the HTML for the Note note type. In the generated `share/panels/NoteDetail/EditNote.html` file, add the `-readonly` option to the LastModBy and LastModDt field substitutions as follows:

```
<TR>
  <TH align=right ><!-- SYNC prompt LastModDt --></TH>
  <TD>
```

```
    <!-- SYNC field LastModDt -readonly -->
  </TD>
</TR>

<TR>
  <TH align=right ><!-- SYNC prompt LastModBy --></TH>
  <TD>
    <!-- SYNC field LastModBy -readonly -->
  </TD>
</TR>
```

The read-only fields display as follows:



Last Modified on	2004-02-11
Last Modified by	Jack Blaine

Example 2: PostSchema Procedure inserting text

This example adds a text message to the note body before updating the note.

```
proc Process_Note_PostSchema {} {
   global SYNC_Parm
   set  prefix "Message appears before append"
   set SYNC_Parm(Appendage) "${prefix}\n$SYNC_Parm(Appendage)"
}
```

Note: The Appendage field is a virtual note field. It is not a field available for display in the notetype, as, for example, Author and Body are. The contents of the Appendage field are stored temporarily while the note is being edited then committed to the Body field along with the note; but the Appendage field can be used to query the note body content before it is committed in order to manipulate the resulting content.

Working with Global Variables in Process Procedures

There are three sets of variables relevant to Process procedures:

- The previous value of a field

  The previous value of a field is the value at the time the field was displayed for editing. These values are stored in the `SYNC_Parm` array. The index into the `SYNC_Parm` array for a field's previous value is `_OLDVALUE_<fieldname>`.

- The current value of a field

The current value of a field is the value at the time the note was submitted. The current value of a field is stored in the `field_values` array. The index into the field_values array is the field name. Access the `field_values` array to test or verify values the user has entered. The `field_values` array is available only to PostSchema procedures, not PreSchema procedures.

Note: Do not modify the `field_values` array; to modify a field value, modify the `SYNC_Parm` array instead.

- The new value of a field

  Store the new value of a field in the `SYNC_Parm` array, indexed by its field name. See Example: PostSchema Procedure to see how to set a new value for a field. Although you access a field's previous value (using `SYNC_Parm(_OLDVALUE_<fieldname>)`) and its current value (using `field_values(<fieldname>)`) for reference purposes, you make changes to the field by setting new field values in the `SYNC_Parm` array.

Examples of these variables are included below.

Example: Displaying Parameter Values

To display the values of the available `SYNC_Parm` variables, include the following code segment in your `Process_<NoteType>_PreSchema` or `Process_<NoteType>_PostSchema` procedure.

```
foreach f [array names SYNC_Parm] {
  puts "$f = $SYNC_Parm($f)<br>"
}
```

This code segment added to the `Process_Note_PostSchema` procedure displays the following results:

```
ProjectConfig = Rel1.1
CCList = jblaine karen
NoteSystem = SyncNotes
_OLDVALUE_Title = New note
Appendage = Started working on ms134 bug.
_SYNCDate_LastModDt = 2004-02-11
_OLDVALUE_LastModDt = 2004-02-11
NoteType = Note
ProjectName = Triton
_OLDVALUE_LastModBy = jblaine
Title = Fix bug in submodule ms134
NoteId = 1
LastModDt = 2004-02-11
```

```
_OLDVALUE_CCList =
LinkedURLs =
LastModBy = karen
command = Process
```

Aborting Note Submission

To abort note submission, add a Tcl `error` command; the note is not submitted:

```
  if {[lsearch -exact $fieldNames LastModBy] == -1} {
      error "Could not find field LastModBy"
  }
```

Tcl Panel Scripts

Implementing Panels Using Tcl Panels Scripts

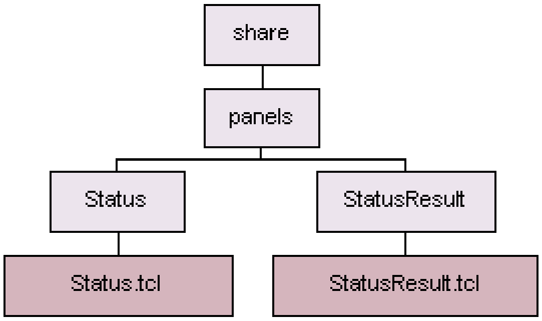
If your panel requires mostly dynamic HTML, you can use a Tcl script to implement the panel. In this case, you create a Tcl script that generates the entire HTML code for the panel. This method of implementing panels is for advanced Tcl programmers.

Implementing an HTML Form in a Tcl Script

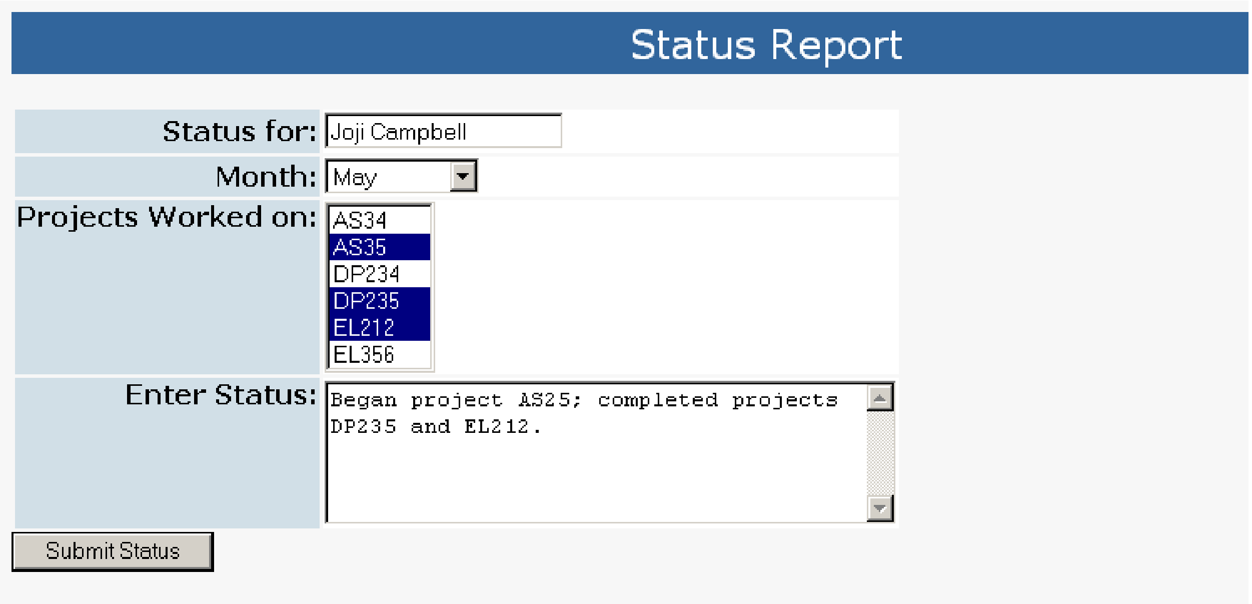
If you create a note panel using a Tcl script, you can use the `call_substitution` utility to generate note fields in a panel. If instead you are creating a custom panel from scratch that is **not** a note panel, you can implement the fields using HTML form input elements. You wrap the HTML tags within `puts` statements to generate your panel.

Example: Creating HTML Forms in a Tcl Script

The `Status.tcl` and `StatusResult.tcl` scripts implement a status report form:



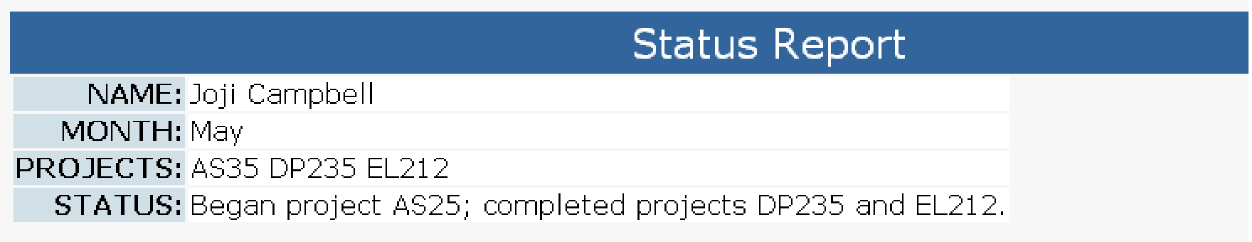
The following sample panel (implemented in `Status.tcl`) uses HTML tags to implement a form that queries for status.



Status Report

Status for:	Joji Campbell
Month:	May
Projects Worked on:	AS34 AS35 DP234 DP235 EL212 EL356
Enter Status:	Began project AS25; completed projects DP235 and EL212.

A second panel (implemented in `StatusResult.tcl`) receives the form input and displays the status results:



Status Report

NAME:	Joji Campbell
MONTH:	May
PROJECTS:	AS35 DP235 EL212
STATUS:	Began project AS25; completed projects DP235 and EL212.

The `Status.tcl` script (below) uses the HTML `<form>` tag to implement a form. The `<select>` tag implements a pull-down Month field, as well as a Projects multiple choice list.

To implement the panel in the style of the ProjectSync cascading style sheets, the `Status.tcl` and `StatusResult.tcl` scripts call the `style` substitution and include the `class=Status` attribute (within the `<body>` HTML tag below).

Status.tcl Script

```
puts {
  <html>
  <head>
  <title>Status</title>
}

call_substitution style
```

```
puts {
  </head>
  <body class="Status">
  <table align="center" width="100%" cellspacing=0 cellpadding=4
border=0>
  <tr><th class="PAGETITLE">Status Report</th></tr>
  </table>

  <form action="/scripts/isynch.dll" method="post">
  <input type="hidden" name="panel" value="StatusResult">

  <table class=LEFTHEADERTBL>
      <tr><th align=left>Status for: </th>
        <td><input type="text" name="name"></td>
      </tr>
      <tr><th align=left>Month:  </th>
        <td><select name="month">
          <option>January
          <option>February
          <option>March
          <option>April
          <option>May
          <option>June
          <option>July
          <option>August
          <option>September
          <option>October
          <option>November
          <option>December
         </select>
      </td>
    </tr>
    <tr><th align=left valign=top>Projects Worked on:</th>
      <td><select name="projects" size="6" multiple>
          <option>AS34
          <option>AS35
          <option>DP234
          <option>DP235
          <option>EL212
          <option>EL356
          </select>
      </td>
    </tr>
    <tr><th align=left valign=top>Enter Status:</th>
      <td><textarea name=report cols=40 rows=5>
        </textarea>
      </td>
    </tr>
    </table>
    <input type="submit" value="Submit Status">
    </form>
  </body>
  </html>
}
```

Example: Passing Control and Arguments to Panels

The `Status.tcl` script (shown in the previous section) passes control and parameters to the `StatusResult.tcl` script (below).

StatusResult.tcl Script

```
cgi_arg name
cgi_arg month
cgi_arg -multi projects
cgi_arg report

puts {
  <html><head><title>StatusResult</title></head>
}

call_substitution style

puts {
  <body CLASS=StatusResult>
  <table align="center" width="100%" cellspacing=0 cellpadding=4
border=0>
  <tr><th class="PAGETITLE">Status Report</th></tr>
  </table>

  <table class=LEFTHEADERTBL>
}

puts "<tr><th align=left>NAME:</th><td>$name</td></tr>"
puts "<tr><th align=left>MONTH:</th><td>$month</td></tr>"
puts "<tr><th align=left>PROJECTS:</th><td>$projects</td></tr>"
puts "<tr><th align=left>STATUS:</th><td>$report</td></tr>"

puts {
  </table></body></html>
}
```

The `Status.tcl` script passes the parameters using the post method, where the browser contacts the form-processing server and sends the data to the server. Note that you set the action attribute of the <form> tag to "/scripts/isynch.dll", the SyncServer process that handles the form:

```
  <form action="/scripts/isynch.dll" method="post">
```

The post method generates a URL that includes the list of parameter name/value pairs and passes the URL to the SyncServer. One of the parameter name/value pairs is the panel parameter specified in the following line of Status.tcl:

```
  <input type="hidden" name="panel" value="StatusResult">
```

This hidden panel parameter indicates that upon submitting the Status panel, the browser must load the StatusResult panel. For this URL panel specification to work, the StatusResult.tcl file must be located in a directory with the same name in a custom /share/panels directory:

```
  /share/panels/StatusResult/StatusResult.tcl
```

The URL includes the rest of the parameter name/value pairs generated by the <input> fields of the Status form. The StatusResult.tcl script uses the cgi_arg utility to access the parameters passed from the Status.tcl script. Because the Projects fields is a multi-valued select list (specified using the multiple attribute of the <select> tag), we use the -multi argument of the cgi_arg utility to access the projects parameter from within the StatusResult.tcl script. See the cgi_arg utility for more information.

Example: Using the format Command to Handle HTML Code

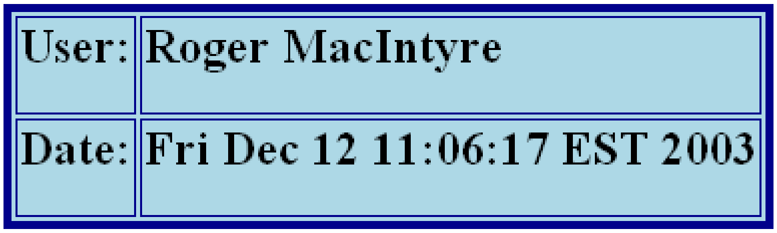
To invoke a proc or substitute a global variable within a chunk of HTML code, you can use the format command within a puts statement to simplify the syntax. The following example uses the format command with the %s string substitution syntax to insert a call to the date proc and to insert the $username variable. The values substituted for the %s strings appear in the last line of the script below.

MyPanel.tcl Script

```
proc getdate {} {
  return [clock format [clock seconds]]
}

proc getuser {} {
  global SYNC_User
  set userUrl sync:///Users/$SYNC_User
  return [url getprop $userUrl Name]
}
```

```
set username [getuser]

puts [format {
  <html>
  <head>
    <title>My Custom Panel</title>
  </head>
  <body>
    <table border=4 bordercolor=darkblue bgcolor=lightblue>
    <tr>
      <td> <h2>User:</h2> </td>
      <td> <h2>%s</h2> </td>
    </tr>
    <tr>
      <td> <h2>Date:</h2> </td>
      <td> <h2>%s</h2> </td>
    </tr>
    </table>
  </body>
  </html>
} $username [getdate]]
```

MyPanel.tcl Results



User:	Roger MacIntyre
Date:	Fri Dec 12 11:06:17 EST 2003

Note Panel Substitution Tags

What Are Substitution Tags?

A substitution is a Tcl procedure that generates dynamic content in a panel. You invoke substitutions inside of a panel's HTML template file. You can also reference substitutions in Tcl panel scripts using the `call_substitution` Tcl utility.

You embed **substitution tags** within HTML panel templates as placeholders for graphical interface elements. The substitution tags appear as HTML comments in the template. For example, a property named Title is implemented with this tag:

```
<!-- SYNC field Title -->
```

Substitution names (in this case, `field`) are case sensitive. A substitution tag generates HTML code for a ProjectSync panel. In most cases, the HTML code generates graphical interface elements in a ProjectSync panel. The placement of the graphical element coincides with the placement of the substitution tag in the HTML template for the panel.

The type of graphical element generated in a note panel by ProjectSync depends on the substitution tag used and its property (field). The type of graphical element generated also depends on which panel is displaying the substitution tag. In the Add and Edit panels, the graphical element displayed is that prescribed for the property type. In the View mode of a note panel, the substitution tag displays as plain text. Also, if a property is access-controlled, the substitution tag displays as plain text.

For example, the `field` substitution tag for the `Title` property generates a type-in field in the Add and Edit modes, but displays as read-only text in View mode. As another example, if a property is of the property type SyncState, a property type built on the primitive StateMachine property type, a cyclic field displays in the Add or Edit panels. The SyncState cyclic field lists the states which are valid transitions from the current state. See ProjectSync User's Guide: Predefined Property Types to learn about the graphical elements generated if you use particular property types.

You can generate the HTML templates for a note type so that you can edit them as described in the Generating HTML Templates and Customizing HTML for Panels topics. The following generated templates correspond to the panels and modes of a note type. These templates are located in the `share/panels` directory of either the site-wide (`<SYNC_SITE_CUSTOM>/share/panels`) or server-specific (`<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/panels`) location.

- `NoteAdd/Add<note_type>.html` - This panel displays if a user selects the **Add** button for the note type. Edit this file to change the layout of the Add panel for the note type.

- `NoteDetail/Edit<note_type>.html` - This panel displays if a user brings up a note of this note type in Edit mode. Edit this file to change the layout of the Edit panel for the note type.
- `NoteDetail/View<note_type>.html` - This panel displays if a user brings up a note of this note type in View mode. Edit this file to change the layout of the View panel for the note type.

If you are adding a field to one of a note type's panel modes such as the `NoteAdd/AddSyncDefect.html` panel, be sure to add the field to its other modes as well (in this case, the `NoteDetail/EditSyncDefect.html` and `NoteDetail/ViewSyncDefect.html` panels).

You use substitution tags in your note panels, as well as in any other ProjectSync panel. Each ProjectSync panel can have its own set of substitutions defined in its initialization (`.ini`) driver file. Likewise, a predefined set of substitutions is accessible to all note panels.

Syntax of Substitution Tags

The descriptions in the following topics provide the syntax of the predefined substitution tags you will find most useful in note panel templates. The tag descriptions include the types of graphical elements the tags generate and the options you can use with them.

The ProjectSync panel substitution tags are implemented as HTML comments with the HTML templates for a note panel, for example:

```
<!-- tagname value -option -->
```

In the following substitution descriptions, the syntax of each substitution is provided with all possible options. Optional arguments are preceded by a hyphen (`-`). You must specify required arguments in the order shown in the syntax descriptions. Specify optional arguments after the required arguments. You can specify optional arguments in any order.

attachment Substitution

```
<!-- SYNC attachment -hidden -rows <#> -cols <#> -->

<!-- SYNC attachment -readonly -rows <#> -cols <#> -->
```

Description

The `attachment` substitution lets users create links in a note to objects of a project. Clicking on an attachment link in a note displays the attached object's data sheet.

Add and Edit Mode

The `attachment` tag generates a type-in field for the Add and Edit modes of note types so that a user can attach a note to a ProjectSync object. The type-in field is defined as a variable named `LinkedURLs`. A **Browse** button displays objects on the SyncServer, so that the user can select objects to attach to the new note. If the Project field is set to a project, the **Browse** button displays the objects defined for that project. If the Configuration field is set to a configuration, the **Browse** button displays the objects defined for that configuration. If a user edits an existing note, the Attach To field displays a list of the current attachments, which the user can edit.

View Mode

For View mode, the `attachment` tag displays as a read-only list of hyperlinked note attachments.

Use

Use of the `attachment` substitution is optional in panel HTML templates.

Arguments

`-readonly` Displays the Attachment values as read-only.

`-hidden` Prevents the Attach To box from displaying on the generated panel. Optional.

`-rows <#>` Controls the height of the text area that displays an editable list of attachments. (Optional) Specify the number of rows as an integer value. Default value is 4. The `-rows` option is not applicable in View mode.

`-cols <#>` Controls the width of the text area that displays an editable list of attachments. (Optional) Specify the number of columns as an integer value. Default value is 40. The `-cols` option is not applicable in View mode.

Globals Referenced

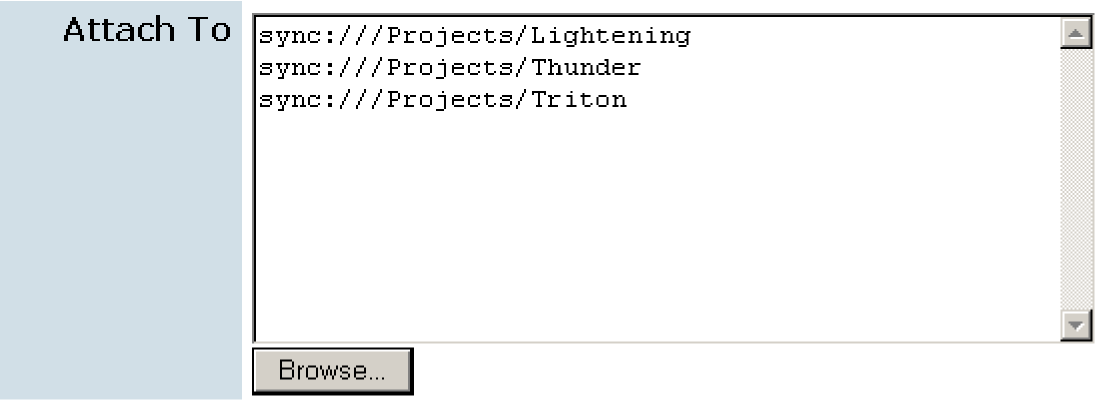
`LinkedURLs` You can seed the Attach To field by invoking the Add mode of the panel using a panel URL with the `LinkedURLs` argument. See Note Panel Arguments for more information.

Examples

```
<TR>
  <!-- SYNC std_header -valign top -mapped ATTACH_TO -->
  <TD>
```

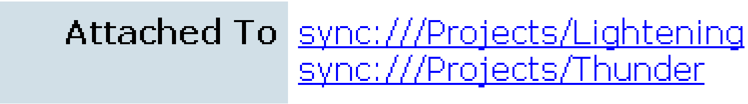
```
    <!-- SYNC attachment -rows 10 -cols 50 -->
  </TD>
</TR>
```

This attachment substitution displays in Add or Edit mode as:



```
sync:///Projects/Lightening
sync:///Projects/Thunder
sync:///Projects/Triton
```

This attachment substitution displays in View mode as:



sync:///Projects/Lightening
sync:///Projects/Thunder

Note that the -rows and -cols options are not meaningful in View mode.

charset Substitution

```
<!-- SYNC charset -->
```

Description

The charset substitution ensures proper display of non-English characters.

Use

Use of the charset substitution is required in all panels, including custom panels that are not note panels.

Arguments

None.

Globals Referenced

None.

Example

```
<!-- SYNC charset -->
```

configuration Substitution

```
<!-- SYNC configuration -hidden -->

<!-- SYNC configuration -readonly -->
```

Description

The `configuration` substitution lets users select a configuration for a project or view the configuration chosen for a project. To seed the Configuration value, see "Note Panel Fields".

Add and Edit Mode

The `configuration` tag generates a pull-down menu that displays a choice list of the configurations for the project selected in the Project cyclic field. If no project is selected or the project has no configurations, the choice list is empty.

If a user edits an existing note, the Configuration cyclic field displays the current value of the configuration.

View Mode

For View mode, the `configuration` tag displays a read-only text box containing the current configuration. The configuration is a hypertext link to the configuration's data sheet.

Use

Use of the `configuration` substitution is optional in panels; however, if there are attachments to a configuration in a note panel and you do not include the `configuration` substitution in each mode of the note panel's HTML templates, the attachment will be deleted during the processing phase. If you do not want to display the Configuration choice list field in one of the panel's modes, use the `-hidden` option of the `configuration` substitution.

Arguments

`-readonly`	Display the Configuration value as read-only. Optional.
`-hidden`	Prevents the Configuration pull-down menu from displaying on the generated panel. Optional.

Globals Referenced

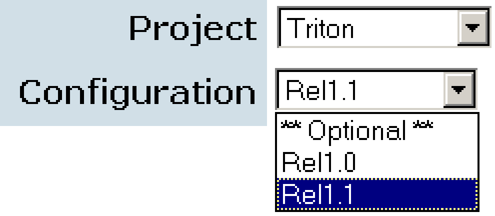
ProjectName — ProjectSync references the `ProjectName` global variable to display the possible configurations that correspond to the project.

ProjectConfig — ProjectSync references the `ProjectConfig` global variable to display the possible configurations that correspond to the project.

Examples

```
<TR>
  <!-- SYNC std_header -mapped CONFIGURATION -->
  <TD>
    <!-- SYNC configuration -->
  </TD>
</TR>
```

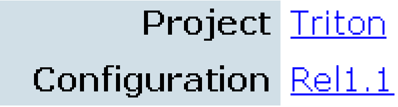
In Add or Edit mode, this `configuration` substitution generates a pull-down menu containing configurations corresponding to the selected project, `Asic`:



Project Triton

Configuration Rel1.1

The `project` and `configuration` substitutions display in View mode as:



Project Triton

Configuration Rel1.1

controls Substitution

```
<!-- SYNC controls -->
```

Description

The `controls` substitution generates the appropriate buttons for a particular note panel. Buttons include:

- **Submit** button - Submits the values the user has set in the note panel and initiates the processing phase.

- **Help** button - Displays the documentation for the current mode of the note type (Add, Edit, or View mode). ProjectSync displays custom documentation if it exists for the note type. See Customizing Documentation for details.
- **Delete** button - Deletes the note being edited. Displays in Edit mode if the user has access rights to delete the note.
- **Show Menu** button - Shows the ProjectSync menu and the Quick View frame. Displays if the user has opened the Edit or View note panel frame in a separate window (and thus the menu and Quick View are not showing).
- **Edit <note type>** button - Changes the note panel from View mode to Edit mode. Displays in View mode.

Add Mode

In the Add mode of a note panel, the `controls` tag generates **Submit** and **Help** buttons.

Edit Mode

In the Edit mode of a note panel, the `controls` tag generates these buttons in the order shown: **Submit**, **Delete** (if the user has access rights to delete the note), **Help**, and **Show Menu** (if the menu and Quick View frames are currently hidden).

View Mode

In the View mode of a note panel, the `controls` tag generates these buttons in the order shown: **Edit <note type>**, **Help**, and **Show Menu** (if the menu and Quick View frames are currently hidden).

Use

Use of the `controls` substitution is required in all panels.

Arguments

None.

Globals Referenced

`field_values`	Referenced to determine whether the user is the author of the note; this information is passed to the `access verify` command to determine if the user can delete the note.
`NoteId`	Passed to `access verify` to determine if the user can delete the note.
`NoteSystem`	Passed to `access verify` to determine if the user can delete the note.

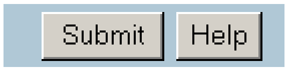
`NoteType` Passed to `access verify` to determine if the user can delete the note.

`SYNC_User` Passed to `access verify` to determine if the user can delete the note.

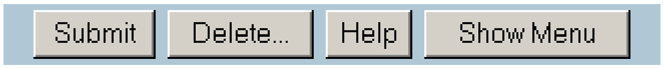
Examples

```
<TR>
  <!-- SYNC std_header -->
  <TD align=center class=CONTROLS>
    <!-- SYNC controls -->
  </TD>
</TR>
```

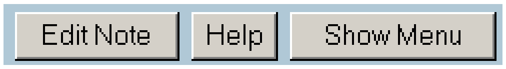
In Add mode, the `controls` substitution can generate these controls:



In Edit mode, the `controls` substitution can generate these controls:



In View mode, the `controls` substitution can generate these controls:



The **Edit** button includes the name of the note type.

eval Substitution

```
<!-- SYNC eval {<Tcl_script>} -->
```

Description

The `eval` substitution executes a block of Tcl code from within an HTML note panel template. Use the `eval` substitution directly in the HTML code instead of having to call a Tcl substitution procedure that you create in a `.ini` initialization driver file.

The `eval` substitution behaves the same in all modes of note panels: Add, Edit, and View.

Use

Use of the `eval` substitution is optional in panels.

Arguments

`{<Tcl_script>}`	A block of Tcl code, enclosed by curly braces ({ }). The code runs in its own scope, so any global variables needed by the code must be explicitly referenced with a global statement. (See example.)

Globals Referenced

None.

Example

This following example displays a greeting directly to the user within the note panel:

```
<!-- SYNC eval {
  global SYNC_User
  puts "<H2>Hello, $SYNC_User.</H2>"
} -->
```

field Substitution

```
<!-- SYNC field <field_name> -choices <choice_list> -cols <#> -
editable -readonly -rows <#> -size <#> -style <style> -values
<value_list> -->
```

Description

The `field` tag generates a type-in field, pull-down menu, check box, radio buttons, or plain text depending on the `field_name` and the mode of the note panel (Add, Edit, or View). The `field_name` must be an existing field defined in the note type using the ProjectSync Note Type Manager. ProjectSync determines the graphical element to generate based on the property type of the field.

A field in a View panel or within an access-controlled note type displays as plain text. The same field within an Add or Edit panel displays as the graphical element corresponding to its property type. To find out the type of element that will be displayed for a particular field, determine the field's property type. For a list of property types and their corresponding graphical elements, see ProjectSync User's Guide: Predefined Property Types.

You can determine the property types for specific fields in an existing note type by bringing up the ProjectSync Note Type Manager and selecting **Modify an existing note type**. Select the note type whose fields you want to review. The required and optional fields display, including their property types and prompts. You can insert the `prompt` substitution in your HTML template to display the prompt for a particular field.

When users view a field in an Add panel, the field displays the default value of the property type. Likewise, when users view the field in an Edit panel, the field is initialized to the current value for the field. If you want a field to be displayed as read-only text within the Edit panel, use the `-readonly` option. Even if you have not set up a `ModifyNoteProperty` access control for the field, it will display as read-only text.

By default, the elements that display in pull-down menus and radio buttons reflect the property type of the field. For example, a pull-down menu of property type SyncPriority contains choices low, medium, high, and stopper, by default. You can change the order or specify a subset of these choices using the `-values` option. You can also change the display choices for these values using the `-choices` option. If you use the `-choices` option, be sure to make corresponding changes to the Add and Edit modes of your note panel. For the `-choices` option, you do not need to update the View mode. If you use the `-values` option, be sure to make corresponding changes to the Add, Edit, and View modes of your note panel. In this case, if you do not update the View mode, the read-only value that displays is the original value from the property type rather than the new value a user might have specified in Add or Edit mode.

The field names you supply can correspond to configurable fields that generate complex fields such as file attachment fields, link notes fields, transcript fields, CC list fields, and keyword fields. See Note Panel Fields: Configurable Fields for details. See the Examples section below for examples of configurable fields, as well.

Use

Use of the `field` substitution is required for each field you want to include in a note panel. Be careful not to include an editable field twice in your template; otherwise, the resulting field value will be unpredictable. You can include the same field twice in Edit mode if one instance of the field is editable and another instance is read-only.

Arguments

`field_name` The name of the field to generate in the note panel. Required for each field in note panels. Include the names of standard fields or configurable fields such as CC list fields, transcript fields, linked notes fields, file attachment fields, and keyword fields. See the Examples section below for examples of fields.

`choice_list` Use the `-choices` option to display alternate choices rather than the predefined choices of the field's property type. Optional. Applies to choice list and state machine fields, as well as Boolean radio button fields. Specify a Tcl list containing the choices. The number of choices must match either the number of choices specified for the field's property type definition or the number of choices specified in the `value_list` you supply with the `-values` option. See Example: Displaying New Values for Choice Lists.

`-cols <#>` For string properties and property types derived from strings (for example, the CCList property type), specifies the number of columns to display for the generated type-in field. Optional. The default value for the number of columns varies, depending on the property type. The unlimited string property type, String, generates a type-in field containing 50 columns by default. A transcript field generates a type-in field containing 40 columns by default. The `-cols` option is ignored for View mode.

`-editable` Used to override the read-only default of View mode. Instead of generating a read-only text box, the `field` substitution with the `-editable` option generates the graphical element corresponding to the field's Edit mode. Optional. For transcript fields, a read-only transcript of previous entries displays, as well as a type-in field for entering new text.

`-readonly` Used to override the default view of the field in Edit mode. Optional. The `-readonly` option is ignored for modes other than Edit.

`-rows <#>` For integer, floating point, and unlimited length string properties, specifies the number of rows to display for the generated type-in field. Optional. The default value for the number of rows varies, depending on the property type. The unlimited string property type, String, generates a type-in field containing 10 rows by default. A transcript field generates a type-in field containing 8 rows by default. The `-rows` option is ignored for View mode.

`-size <#>` For limited length string properties and their derivatives, controls the maximum number of characters allowed in a type-in field in Add and Edit mode. Optional. The default value for the string size varies, depending on the property type.

`-style` For Boolean, state machines, and choice lists, controls the type of graphical element generated in Add and Edit modes. Optional. The default elements for each field depend on the property type. Ignored for View mode.

Valid styles for Booleans are:

- checkbox – For Boolean properties only. Displays a single checkbox where the unchecked state represents an off (or False) value and the checked state represents an on (or True) value. Default object for Boolean property types.
- toggle – Same as checkbox.
- radio – Displays a set of radio buttons, one for each

possible value for the property type, arranged vertically.·

- hradio – Displays a set of radio buttons, one for each possible value for the property type, arranged horizontally.
- vradio – Same as radio.

Valid styles for state machines and choice lists are:

- menu – Displays a pull-down menu of all possible choices for the property. Default object for state machine and choice list property types.
- select – Same as menu.

- radio – Displays a set of radio buttons, one for each possible value for the property type, arranged vertically.·
- hradio – Displays a set of radio buttons, one for each possible value for the property type, arranged horizontally.

- vradio – Same as radio.

`value_list` Use the `-values` option to change the order or specify a subset of the elements in Boolean, choice list, or state machine fields. Optional. Specify a Tcl list containing the values. The `value_list` defaults to the legal set of values for the property type. The `value_list` argument is ignored for View mode. See Example: Displaying New Values for Choice Lists.

Globals Referenced

AllProjectsMap

AllUsersList

AllUsersMap

classes

DisplayMode

field_values

NoteId

NoteSystem

NoteType

ptypes

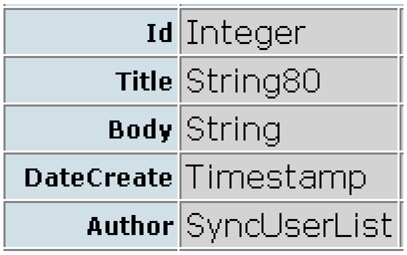
reqfields

SYNC_User

Examples

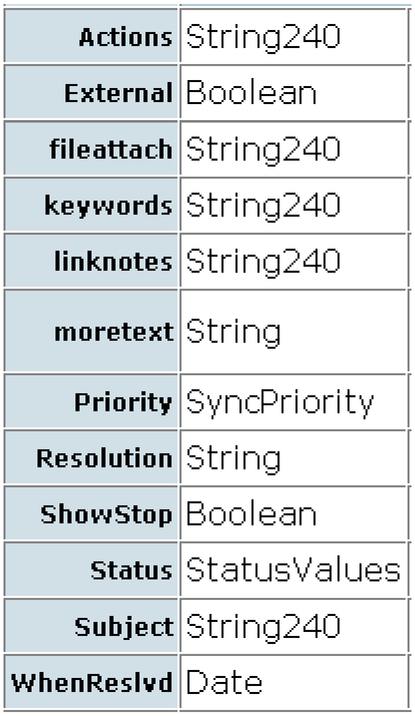
The note type used for the following examples has the fields and associated property types shown below.

Required Fields



Id	Integer
Title	String80
Body	String
DateCreate	Timestamp
Author	SyncUserList

Optional Fields



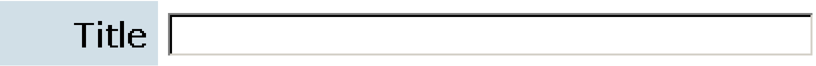
Actions	String240
External	Boolean
fileattach	String240
keywords	String240
linknotes	String240
moretext	String
Priority	SyncPriority
Resolution	String
ShowStop	Boolean
Status	StatusValues
Subject	String240
WhenReslvd	Date

Example: Type-In Field

The Title field shown below has property type String80 and thus generates an 80 character type-in field in Add and Edit mode. The substitution shown below is located in the HTML Add mode template for the note type: `/share/panels/NoteAdd/Add<notetype>.html`.

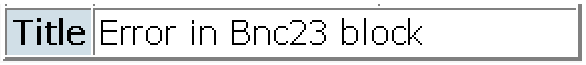
A similar substitution is added for the Edit panel mode in: `/share/panels/NoteDetail/Edit<notetype>.html`.

```
<!-- SYNC field Title -->
```



In View mode, the Title field generates a read-only text box. You can add HTML code to customize the display of the field. The code shown below specifies a white background. The HTML segment shown below is located in the HTML View mode template for the note type: `/share/panels/NoteDetail/View<notetype>.html`.

```
  <TD colspan=2 bgcolor=white>
    <!-- SYNC field Title -->
  </TD>
```

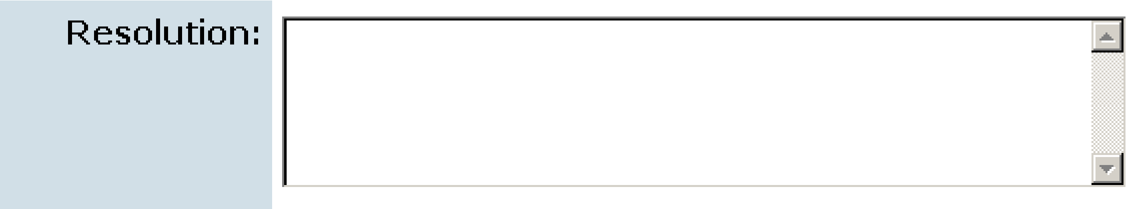


Title Error in Bnc23 block

Example: Multi-Line Type-In Field

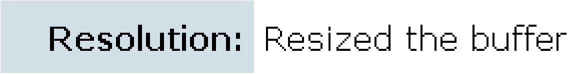
You can specify the number of rows for a multiple line field using the `-rows` option. The field's property type must be of the unlimited length string type (`String`). The following substitution generates the multi-line field shown below in Add mode.

```
<!-- SYNC field Resolution -rows 4 -->
```



The following substitution generates the read-only text box shown below in View mode. The `-rows` option is not valid in View mode so it is not included; if included for View mode, the `-rows` option is ignored.

```
<!-- SYNC field Resolution -->
```

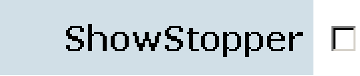


Resolution: Resized the buffer

Example: Check Box Field

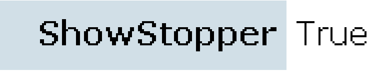
By default, the ShowStop Boolean field generates a check box field in Add mode, as shown below. In Edit mode, the same check box displays with the box checked if the current value of the field is true.

```
<!-- SYNC field ShowStop -->
```



ShowStopper □

In View mode, the ShowStop field generates plain text containing the Boolean value of the field:

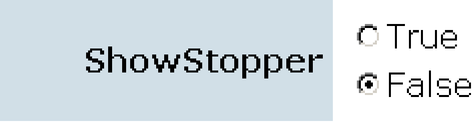


ShowStopper True

Example: Radio Button Field

Radio buttons are supported for Boolean, state machine, and choice list fields. The ShowStop Boolean field, with the vradio style applied below, generates vertical radio buttons in Add and Edit mode.

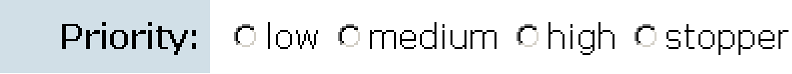
```
<!-- SYNC field ShowStop -style vradio -->
```



ShowStopper
○ True
● False

The Priority field, a state machine of property type SyncPriority, generates a row of radio buttons for Add and Edit modes with the radio style shown below. You specify the default value for the field when you create it using the ProjectSync Note Type Manager.

```
<!-- SYNC field Priority -style radio -->
```



Priority: ○ low ○ medium ○ high ○ stopper

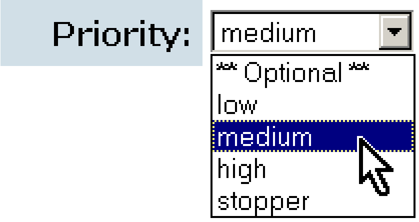
Example: Pull-Down Menu

ProjectSync generates a pull-down menu in Add or Edit mode if a field's property type is a state machine or a choice list. ProjectSync provides predefined property types that generate pull-down menus including SyncClass, SyncPriority, SyncState, and

SyncUserList. See ProjectSync Help: Predefined Property Types to see the choices included in the pull-down menus generated for these property types.

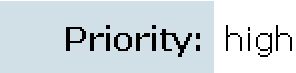
The Priority field shown below is of property type, SyncPriority. In Add and Edit mode, a SyncPriority field generates a pull-down menu with a choice list of low, medium, high, and stopper.

```
<!-- SYNC field Priority -->
```



Priority: medium

The same substitution in View mode generates a text box that displays the current setting for the Priority field:



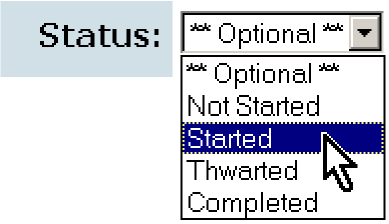
Priority: high

Example: Customized Pull-Down Menu

To create custom pull-down menus with your own choices, you must first design a new property type that includes the choices. You can create a property type that is a state machine, in which the choices presented to the user depend on the current value of the pull-down menu. Or you can create a property type that is a simple choice list, the order of which you define. After you create the new property type, you can add a field of this property type to your note type definition. You can perform both these tasks using the ProjectSync Note Type Manager. See ProjectSync User's Guide: Creating New Property Types and ProjectSync Help: Creating Custom Note Types for details.

The Status field is of a custom property type, StatusValues, which has four choices: Not Started, Started, Thwarted, and Done:

```
<!-- SYNC field Status -->
```



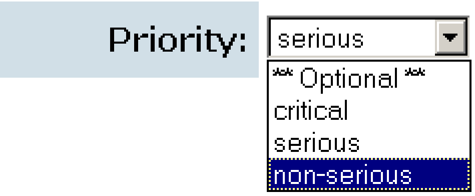
Status: ** Optional **

Example: Displaying New Values for Choice Lists

The following example shows how to change the choices in a pull-down menu. You can change the choice list values by updating the note type or you can make changes in your HTML note type templates as shown below. This example changes the values, order, and display choices of the Priority field, of property type SyncPriority. SyncPriority has values low, medium, high, and stopper, in that order. To create a subset of these values and change their order, this example uses the `-values` option. To display new choices for these values, the example uses the `-choices` option:

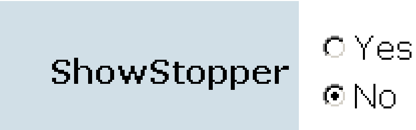
```
<!-- SYNC field Priority -values {high medium low}
  -choices {critical serious non-critical} -->
```

The pull-down menu that displays in Add and Edit mode reflects these changes:



You can also change the choices that display in Boolean radio button fields. By default, Boolean values display in radio buttons as True and False.

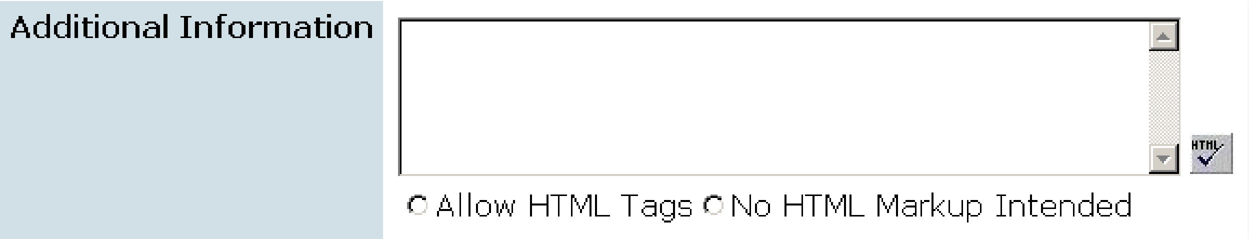
```
<!-- SYNC field ShowStop -style vradio -choices {Yes No}  -->
```



Example: Transcript Field

The `moretext` field in the HTML segment below is a transcript field:

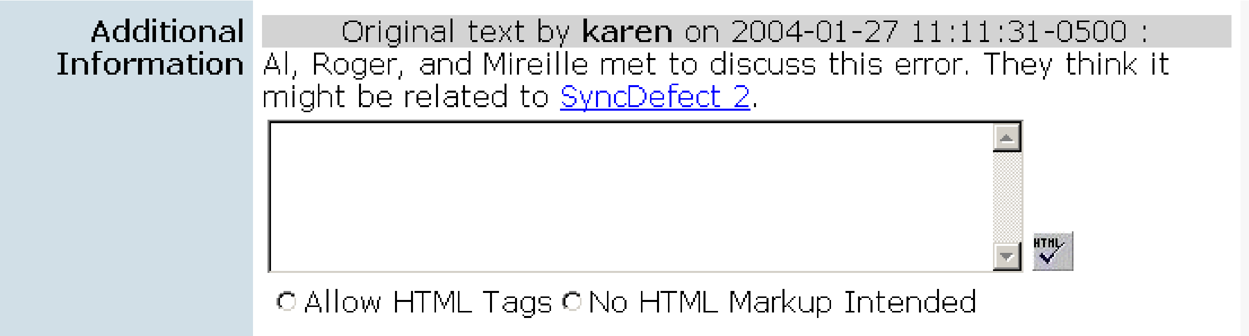
```
<TR>
  <TH align=right valign=top>
    <!-- SYNC prompt moretext -->
  </TH>
  <TD>
    <!-- SYNC field moretext -->
  </TD>
</TR>
```



Additional Information

○ Allow HTML Tags ○ No HTML Markup Intended

In Edit mode, the `moretext transcript` substitution generates a transcript of all prior entries, annotated with each author and date, followed by a type-in field:



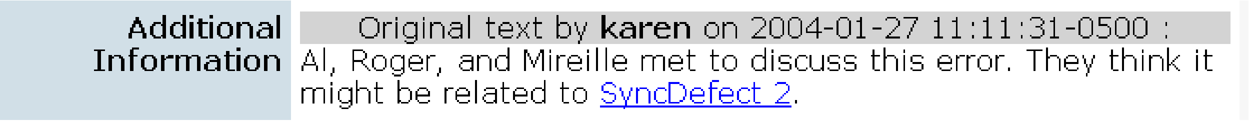
Additional Information

Original text by **karen** on 2004-01-27 11:11:31-0500 :

Al, Roger, and Mireille met to discuss this error. They think it might be related to SyncDefect 2.

○ Allow HTML Tags ○ No HTML Markup Intended

In View mode, the `moretext transcript` substitution generates a transcript:



Additional Information

Original text by **karen** on 2004-01-27 11:11:31-0500 :

Al, Roger, and Mireille met to discuss this error. They think it might be related to SyncDefect 2.

You can create a transcript field by either naming the field `transcript` when you create your note type or by mapping the field name to the `transcript` keyword in the note type configuration file (`/share/config/<notetype>.conf`). The following line from the `<notetype>.conf` file maps the moretext field to the `transcript` keyword:

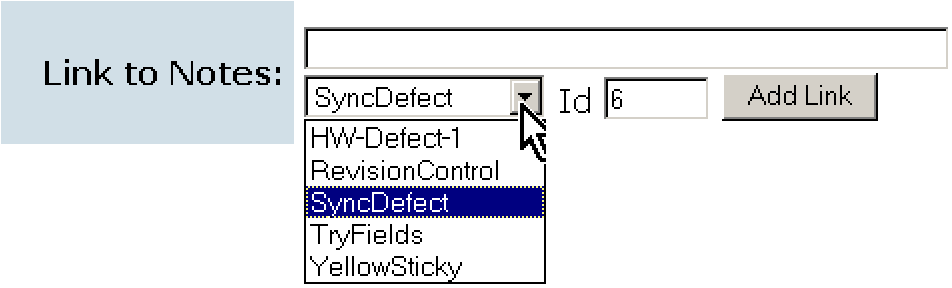
```
set classes(moretext) transcript
```

The `moretext` field is mapped to `transcript`, and like all transcript fields, its property type is the unlimited String type. When the `moretext` field is substituted as shown in the HTML segment above, the Add panel generates a type-in field. In the note type definition, the `moretext` prompt is set to `Additional Information`; thus, this prompt displays beside the type-in field. As with all transcript fields, radio buttons **Allow HTML Tags** and **No HTML Markup Intended** follow the field to let the user specify whether the transcript includes HTML tags that should be interpreted in the resulting text box.

Example: Link Notes Field

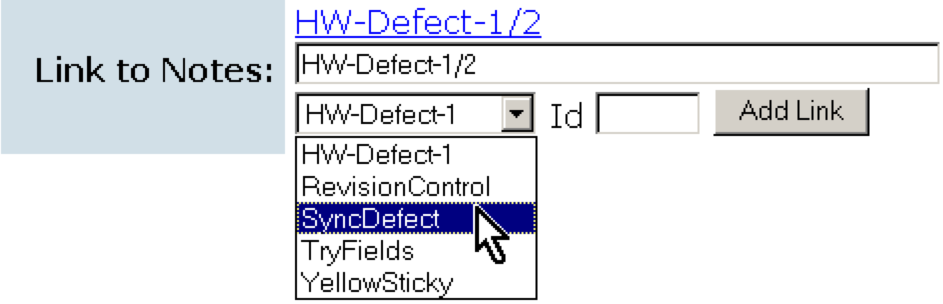
A link notes field provides an easy way for a user to create hypertext links to existing notes on the SyncServer. The `linknotes` field substitution, shown below in Add mode, generates a pull-down menu containing the note types available on the server, a type-in field for the Id number of the note, and an **Add Link** button. After the user selects the note type, enters the Id number, and presses the **Add Link** button, the note type and Id display in the type-in field. You can use the `prompt` substitution to display the prompt, in this case "Link to Notes:". The prompt is defined when you create the fields of the note type using the ProjectSync Note Type Manager.

```
<!-- SYNC field linknotes -->
```

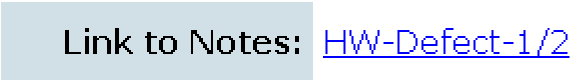


The link notes field is a configurable field; if you need to create more than one, you do so in the `<notetype>.conf` file. See Note Panel Fields: Configurable Fields for details.

Following is the link notes field in Edit mode. The hypertext links display in Edit mode so that you can go to the linked notes, as well as enter new links in Edit mode.



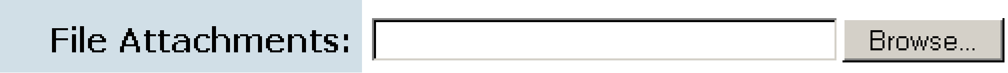
Following is the link notes field in View mode:



Example: File Attachment Field

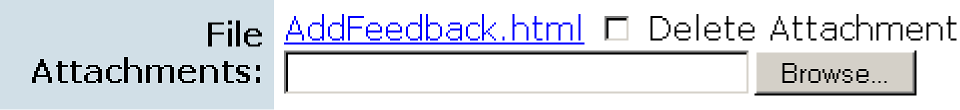
A file attachment field lets users create hypertext links to a file on a server. The `fileattach` field substitution, shown below in Add mode, generates a type-in field for the file links and a **Browse** button. Users browse their directories to select a file to be uploaded to the SyncServer; the resultant link to the file displays in the type-in field. You can use the `prompt` substitution to display the prompt, in this case "File Attachments:". The prompt is defined when you created the fields of the note type.

```
<!-- SYNC field fileattach -->
```

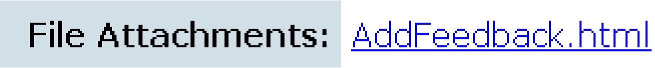


The link notes field is a configurable field; if you need to create more than one, you do so in the `<notetype>.conf` file. See Note Panel Fields: Configurable Fields for details.

The Edit mode provides a list of file links so that a user can bring up a linked file, as well as add a new file attachment. The Edit mode of a file attachment field also generates a check box so that users can delete an attachment.



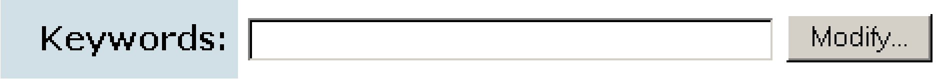
The View mode provides the list of file links:



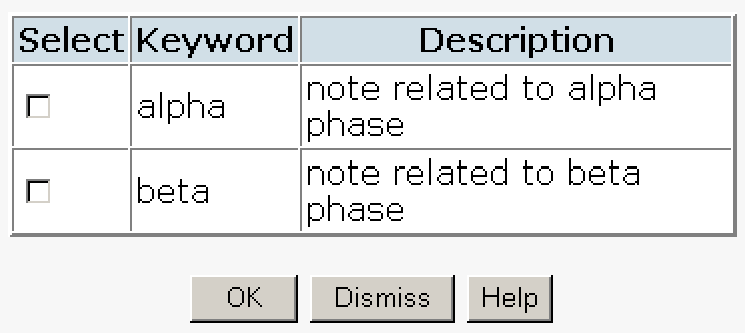
Example: Keywords Field

Another type of configurable field is a Keywords field. You register keywords using the Keyword Manager, available in ProjectSync's Note Type Manager. The `keywords` field substitution, shown below in Add mode, generates a type-in field for the keywords and a **Modify** button.

```
<!-- SYNC field keywords-->
```

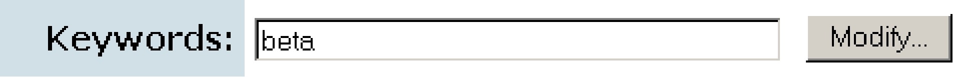


The **Modify** button displays a Keyword Manager list box that displays the existing keywords and their meanings, as well as a check box so that users can select a keyword to be added to the type-in field:



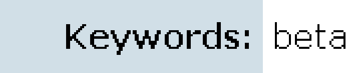
Select	Keyword	Description
□	alpha	note related to alpha phase
□	beta	note related to beta phase

Edit mode provides a list of keywords and the **Modify** button:



Keywords: beta

View mode provides a read-only list of keywords:



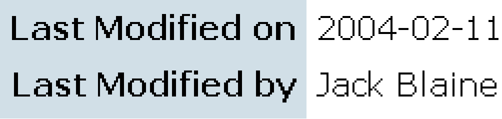
Keywords: beta

Example: Read-Only Field

Use the -readonly option to generate read-only fields using this syntax:

```
<!-- SYNC field LastModDt -readonly -->
....
<!-- SYNC field LastModBy -readonly -->
```

The read-only fields display as follows:



Last Modified on	2004-02-11
Last Modified by	Jack Blaine

hiddens Substitution

```
<!-- SYNC hiddens -->
```

Description

The `hiddens` substitution generates a set of hidden parameters necessary for the proper operation of the various modes of a panel -- Add, Edit, and View mode. For example, by using hidden substitutions, you can pass parameters from the Add mode of a panel to the Process phase of the panel.

Use

Use of the `hiddens` substitution is required in all panels.

Arguments

None.

Globals Referenced

classes

field_values

fields

NoteId

NoteSystem

NoteType

Example

The following `hiddens` substitution generates the hidden fields shown below in the HTML source code of a note panel:

```
<!-- SYNC hiddens -->

<INPUT type=hidden name=NoteSystem value=SyncNotes>
<INPUT type=hidden name=panel value=NoteAdd>
<INPUT type=hidden name=command value=Process>
<INPUT type=hidden name=NoteType value='Note'>
```

See Example: Passing Control and Arguments to Panels in the topic, Implementing Panels Using Tcl Panel Scripts for a complete example.

js_data Substitution

```
<!-- SYNC js_data -->
```

Description

The `js_data` substitution generates a set of javascript data necessary for the proper operation of the graphical elements and various modes of a panel -- Add, Edit, and View mode.

Use

Use of the `js_data` substitution is required in all panels.

Arguments

None.

Globals Referenced

AllProjectsList

AllProjectsMap

AllUsersList

classes

reqfields

NoteForm Substitution

```
<!-- SYNC NoteForm -->
```

Description

The `NoteForm` substitution generates the proper `FORM` HTML tag to implement the note panel.

Use

Use of the `NoteForm` substitution is required for note panel HTML templates.

Note: If you are modifying an HTML panel template and your note panel contains a file attachment field, you must include a `NoteForm` substitution. The `NoteForm` substitution generates an HTML `FORM` tag with an attribute that ensures the proper operation of the file attachment fields. If you add a file attachment field when you create your note type using the ProjectSync Note Type Manager, ProjectSync adds the `NoteForm` substitution automatically. If you are implementing a custom panel that is not a ProjectSync note panel or if you want to include your own validate function within the `FORM` tag, you must include your own `FORM` tag with the `enctype` attribute shown below in bold typeface:

```
<FORM name=NoteAddForm action=isynch.dll enctype=multipart/form-
data method=post onSubmit='return Validate(this)'>
```

Use the `NoteForm` substitution in all note panels even if your note panel does not include a file attachment.

Arguments

None.

Globals Referenced

None.

NoteId Substitution

```
<!-- SYNC NoteId -->
```

Description

The `NoteId` substitution retrieves the Id value of the note being displayed or edited.

Use

Use of the `NoteForm` substitution is optional for note panels.

Arguments

None.

Globals Referenced

NoteId

Example

The following substitution generates the Note Id value:

```
<CENTER><H2>Note #<!-- SYNC NoteId --></H2></CENTER>
```

Note #1

notetype Substitution

```
<!-- SYNC notetype -->
```

Description

The `notetype` substitution retrieves the note type of the note being displayed or edited.

Use

Use of the `notetype` substitution is optional for note panels.

Arguments

None.

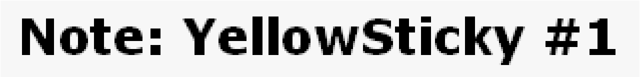
Globals Referenced

NoteType

Example

The following substitution generates the note type and note Id:

```
<CENTER><H2>Note: <!-- SYNC notetype -->#
<!-- SYNC NoteId --></H2></CENTER>
```



Note: YellowSticky #1

pagetitle Substitution

```
<!-- SYNC pagetitle -->
```

Description

The `pagetitle` substitution returns the display mode and the note type of the note. If the substitution is included in Edit or View mode, it also returns the note ID of the note being edited or displayed.

Use

Use of the `pagetitle` substitution is optional for note panels.

Arguments

None.

Globals Referenced

DisplayMode

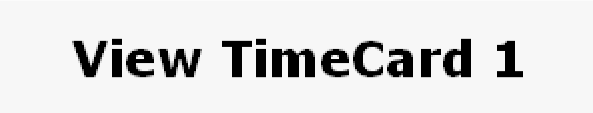
NoteType

NoteId

Example

The following substitution in `ViewTimeCard.html` generates the title below:

```
<CENTER><H2><!-- SYNC pagetitle --></H2></CENTER>
```



View TimeCard 1

project Substitution

```
<!-- SYNC project -hidden -->
```

```
<!-- SYNC project -readonly -->
```

Description

The `project` substitution generates the controls for setting or viewing the project for a note. To seed the Project value, see "Note Panel Fields".

Add and Edit Mode

The `project` tag generates a pull-down menu that displays a choice list of the projects defined on the SyncServer. If a user edits an existing note, the Project pull-down menu displays the current project.

View Mode

For View mode, the `project` tag displays a read-only text box containing the current project. The project name is a hypertext link to the project's data sheet.

Use

Use of the `project` substitution is required in panel HTML templates.

Arguments

`-readonly` Display the Project value as read-only. Optional.

`-hidden` Prevents the Project pull-down menu from displaying on the generated panel. Optional.

Globals Referenced

ProjectName

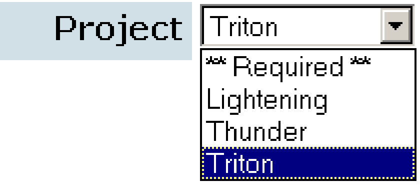
DisplayMode

project

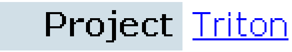
Examples

```
<TR>
  <!-- SYNC std_header -mapped PROJECT -->
  <TD>
    <!-- SYNC project -->
  </TD>
</TR>
```

In Add or Edit mode, this `project` substitution generates a pull-down menu containing the projects on the SyncServer:



The `project` substitution displays in View mode as:



prompt Substitution

```
<!-- SYNC prompt <field_name> -->
```

Description

The `prompt` substitution displays the prompt for a field. You specify the prompt for the field when you create it using the ProjectSync Note Type Manager.

Use

Include the prompt substitution for each field and for each mode of a note type.

Arguments

`field_name` The name of the field whose prompt you wish to display.

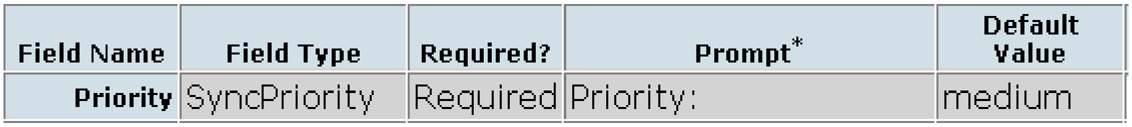
Globals Referenced

ProjectName

ProjectConfig

Example

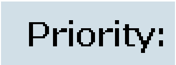
The Priority field is defined in its note type as follows:



Field Name	Field Type	Required?	Prompt*	Default Value
Priority	SyncPriority	Required	Priority:	medium

Thus, the following HTML segment with the prompt substitution generates the prompt displayed below.

```
<TH align=right>
  <!-- SYNC prompt Priority -->
</TH>
```



Priority:

scripts Substitution

```
<!-- SYNC scripts -->
```

Description

The `scripts` substitution generates a set of JavaScript data necessary for the proper operation of the graphical elements and various modes of a panel -- Add, Edit, and View mode.

Use

Use of the `scripts` substitution is required in all panels.

Arguments

None.

Globals Referenced

DisplayMode

Example

<!-- SYNC scripts -->

std_header Substitution

```
<!-- SYNC std_header <head_text> <head_args> -prompt
<field_name> -mapped <term> -head_color <color> -valign
<align_value> -hidden -->
```

Description

The `std_header` substitution inserts a consistent display for a read-only table cell. Generally, you use the `std_header` substitution to label a field. You can enter your own table header as a string or use the `-prompt` option to access the name of the field you specified when you created the note type. You can also insert a term from a ProjectSync string table by using the `-mapped` option. See String Table Configuration Files for more information.

Use

Use of the `std_header` substitution is optional in panels.

Arguments

`<head_text>`	A string of text to display. Optional. Defaults to "".
`<head_args>`	A string containing HTML attribute name/value pairs applicable to the table head tag. See an HTML language reference for valid table attributes. Optional. Defaults to "".
`-prompt <field_name>`	Displays the prompt corresponding to the field name from the ProjectSync string table. Optional.
`-mapped <term>`	Displays the text corresponding to the specified term in the ProjectSync string table. Optional.
`-head_color <color>`	Sets the color of the text displayed. Specify the color using a standard color name or its red, green, and blue (RGB) components, for example #008080. See an HTML language reference for standard colors and RGB values.
`-valign <align_value>`	Controls the vertical alignment of text displayed. Optional. Defaults to middle. Valid values are top, middle or bottom.

`-readonly` Displays the header as read-only. Use the `-readonly` option if you are also using the `-readonly` option with the project, configuration or attachment substitution to display a corresponding project, configuration or attachment as read-only. Optional.

`-hidden` Prevents the header from displaying. Use the `-hidden` option if you are also using the `-hidden` option with the project, configuration or attachment substitution to prevent a corresponding project, configuration or attachment from displaying. Optional.

Notes:

The arguments `-readonly` and `-hidden` are mutually exclusive. If you use both with one field that will generate an error.

You should avoid making the Configuration field hidden or readonly while leaving the Project field unrestricted and editable, since changes to the Project field would not be accurately reflected in the Configuration field.

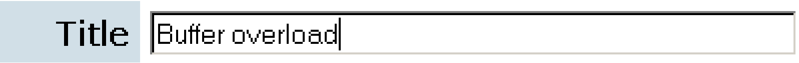
Globals Referenced

None.

Examples

In the following example, ProjectSync displays the prompt for the Title field:

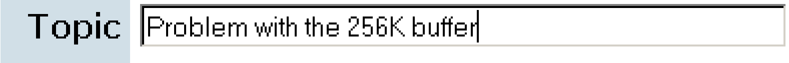
```
<TR>
  <!-- SYNC std_header -prompt Title -->
  <TD>
    <!-- SYNC field Title -->
  </TD>
</TR>
```



Title Buffer overload

In this example, a new text string is provided for the field's label (corresponding to the `<head_text>` argument):

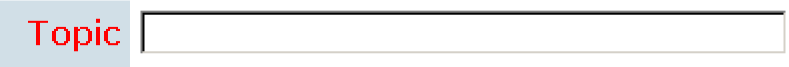
```
<TR>
  <!-- SYNC std_header "Topic" -->
  <TD>
    <!-- SYNC field Subject -->
  </TD>
</TR>
```



Topic Problem with the 256K buffer

In this example, new colors are specified for the text and the background:

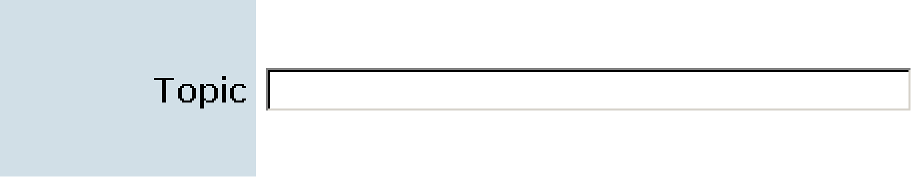
```
<!-- SYNC std_header "Topic" -head_color "red" -->
```



Topic

The following example uses an HTML attribute/value pair as the optional `<head_args>` argument to increase the height of the table cell:

```
<!-- SYNC std_header "Topic" "height=80" -->
```



Topic

std_separator Substitution

```
<!-- SYNC std_separator <separator_text> -tdclass SEPARATOR -
mapped <term> -->
```

Description

The `std_separator` substitution generates a separator in the existing ProjectSync display style. Generally, you use the `std_separator` substitution as a heading for a group of related fields. You can enter your own heading as the separator text or use the `-mapped` option to specify a string already stored in the ProjectSync string table. See String Table Configuration Files for more information. The generated separator is a fully formatted table row with two columns. The left column is empty and the right column contains the specified text.

Use

Use of the `std_separator` substitution is optional in panels.

Arguments

`<separator_text>` A string of text to display. Optional. Defaults to “”.

`-tdclass SEPARATOR` Specifies how the separator displays by setting the table subclass to SEPARATOR. The default ProjectSync cascading style sheet (`.css`) file defines how this subclass displays. See the default ProjectSync style sheets in the `<SYNC_DIR>/share/content/css` directory.

`-mapped <term>` Displays the text corresponding to the specified term in the ProjectSync string table. Optional.

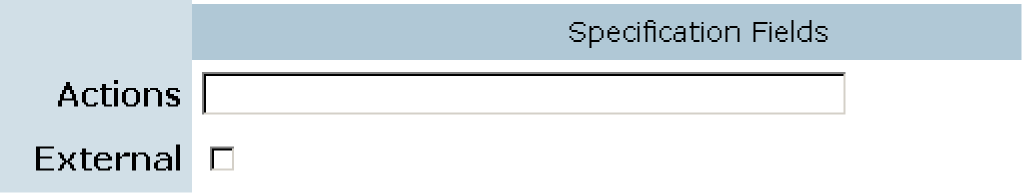
Globals Referenced

None.

Example

The following example shows how to generate a separator:

```
<!-- SYNC std_separator "Specification Fields" -->
```



style Substitution

```
<!-- SYNC style -->
```

Description

The `style` substitution is required for your panel to adhere to the ProjectSync cascading style sheets.

Use

Use of the `style` substitution is required in all panels.

Arguments

None.

Globals Referenced

None.

Example

In the example below, the substitutions and tags in bold cause the panel to display according to the attributes of the ProjectSync cascading style sheets (located at `$SYNC_DIR/share/content/css`). ProjectSync adds these substitutions and tags automatically when you create a note type:

```
<HTML>
<HEAD>
  <!-- SYNC charset -->
  <!-- SYNC style -->
  <TITLE>%(pagetitle -banner)%</TITLE>
  <!-- SYNC scripts -->
  <!-- SYNC js_data -->
</HEAD>

<BODY class=NoteTemplate onLoad="onLoadCB()"
onUnload="onUnloadCB()">

<CENTER>
  <TABLE align=center width=100% cellspacing=0 cellpadding=4
border=0>
    <tr><th class="PAGETITLE">%(pagetitle)%</TH></TR>
  </TABLE>
  <BR>
</CENTER>

<!-- SYNC NoteForm -->
<!-- SYNC hiddens -->

<TABLE border=0 cellpadding=5 cellspacing=0 rows=2
class=LEFTHEADERTBL>
<!-- SYNC std_separator -tdclass SEPARATOR -mapped OBJ_ATTACH --
>

<TR>
<!-- SYNC std_header -mapped PROJECT -->
  <TD>
    <!-- SYNC project -->
  </TD>
</TR>

<TR>
<!-- SYNC std_header -mapped CONFIGURATION -->
  <TD>
    <!-- SYNC configuration -->
  </TD>
</TR>

<TR>
<!-- SYNC std_header -valign top -mapped ATTACH_TO -->
  <TD>
    <!-- SYNC attachment -->
  </TD>
</TR>

<!-- SYNC std_separator -tdclass SEPARATOR -mapped
REQUIRED_ATTRS -->

<TR>
<!-- SYNC std_header -prompt Title -->
  <TD>
    <!-- SYNC field Title -->
  </TD>
</TR>

<TR>
  <TH align=right  bgcolor=#CCDDD0>
    <!-- SYNC prompt Project -->
  </TH>
  <TD>
    <!-- SYNC field Project -->
  </TD>
</TR>

...
...
...

<TR>
<!-- SYNC std_header -->
  <TD align=center class=CONTROLS>
    <!-- SYNC controls -->
  </TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Note Panel Global Variables

Global Variables

Global variables for ProjectSync are the set of all variables defined at the global scope. The global variables described in this section are available to all note panels. Most of these variables are accessible from all panel modes -- Add, Edit, and View modes.

Each panel declares its own set of global variables required to carry out its functionality. In most cases the global variables that one panel sets up have no bearing on the operation of another panel; do not modify these global variables directly. Use note panel arguments to pass values to panels. See Note Panel Arguments for more information.

Tips:

- To view the value of each global variable, introduce a syntax error in a panel's HTML code. The resulting diagnostic panel displays the values of all global variables.
- Use the `var_substitution` utility in your scripts or `.ini` initialization files as a shorthand for outputting the value of a global Tcl variable.

AllModulesList

The `AllModulesList` global variable provides a list of all modules. The `AllModulessList` global variable is available in Add and Edit mode, but not in View mode.

Type

The `AllModulesList` global variable is a list of [category>]/module names, sorted by category first, if applicable, then module name.

Format

`AllModulesList` has the following format:

```
{[category]module1 [category]module2 [category]module3 ...}
```

Example

Example: Listing AllModulesList

Typing the following command in the Edit <notetpe>.ini file:

```
“puts $AllModulesList”
```

Might product an output such as the following which shows four modules, one of which, cpu, is contained in the "chips" category.:

```
Apex chips/CPU Phoenix Yuma
```

AllProjectsList

The `AllProjectsList` global variable provides a list of all top-level projects and their attributes, including configuration information. The `AllProjectsList` global variable is available in Add and Edit mode, but not in View mode.

Type

The `AllProjectsList` global variable is a list, sorted by project name.

Format

`AllProjectsList` has the following format:

```
{project1 {project1_attributes} project2 {project2_attributes}
project3 {project3_attributes} ...}
```

Each project on the SyncServer appears in the list in this format:

```
<project> {Owner <userid> [Configs <{configlist}>]}
```

`<project>` The project name. Required.

`Owner <userid>` The `Owner` identifier is followed by the owner's user ID. Required.

`Configs <{configlist}>` The optional `Configs` identifier is followed by a list of configurations in this format:

```
{config1 {config1_attributes} config2
{config2_attributes} ...}
```

Examples

Example: Listing AllProjectsList

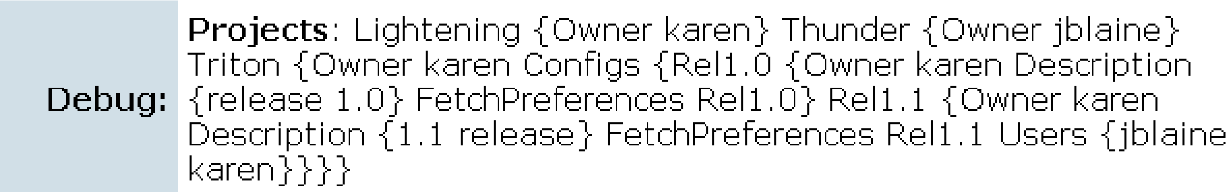
The following substitution, defined in a custom panel initialization file, `share/panels/NoteDetail/EditTimecard.ini`, prints the `AllProjectsList` variable:

```
substitution dumpproject {-allprojects} {
  global project AllProjectsList

  if {$allprojects == 0} {
    puts "<B>Project</B>: $ProjectName<BR>"
  } else {
    puts "<B>Projects</B>: $AllProjectsList<BR>"
  }
}
```

The substitution is included in the `EditTimecard.html` template as follows:

```
<TR>
  <!-- SYNC std_header "Debug:" -->
  <TD align=center>
    <!-- SYNC dumpproject -allprojects -->
  </TD>
</TR>
```

The resulting panel displays:



Debug: **Projects**: Lightening {Owner karen} Thunder {Owner jblaine} Triton {Owner karen Configs {Rel1.0 {Owner karen Description {release 1.0} FetchPreferences Rel1.0} Rel1.1 {Owner karen Description {1.1 release} FetchPreferences Rel1.1 Users {jblaine karen}}}}

The `AllProjectsList` variable lists all the projects on the SyncServer. For example, the third project is Triton (formatting added to make list readable):

```
Triton {Owner karen Configs {
  Rel1.0 {Owner karen Description {release 1.0}
          FetchPreferences Rel1.0}
  Rel1.1 {Owner karen Description {1.1 release}
          FetchPreferences Rel1.1 Users {jblaine karen}}}}
```

The `Triton` project has owner `karen` and two configurations, `Rel1.0` and `Rel1.1`.

Example: Listing Only the Projects in AllProjectsList

The following example separates the projects in `AllProjectsList` from the attributes, storing the project names in a list named `proj` and the attributes (owner and optional configurations) in a list named `properties`. The substitution then lists just the projects:

```
substitution listprojects {} {
  global AllProjectsList

  puts "Select a project: <br>"
  foreach {proj properties} $AllProjectsList {
    puts "Project: $proj<br>"
  }
}
```

AllProjectsMap

The `AllProjectsMap` global variable provides an array containing all top-level projects and their attributes, including configuration information. The `AllProjectsMap` global variable is available in Add and Edit mode, but not in View mode.

Type

The `AllProjectsMap` global variable is an array, indexed by project name.

Format

The format is identical to the `AllProjectsList` variable, with the exception that the project names are used as the array indices. An array index value contains the project attribute list. See `AllProjectsList` for the contents of the projects list.

Example

The following substitution, defined in a custom panel initialization file, `share/panels/NoteDetail/EditTimecard.ini`, prints the attribute list corresponding to the current project by indexing the `AllProjectsMap` variable:

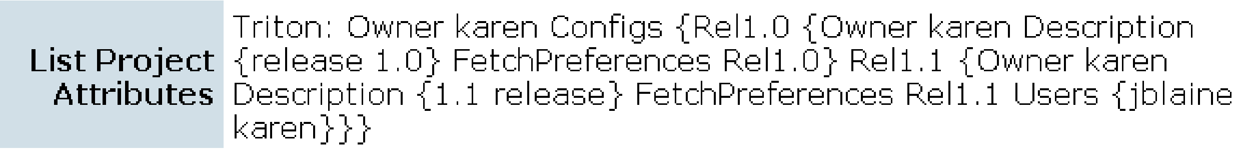
```
substitution listattributes {} {
  global ProjectName
  global AllProjectsMap

  set leafproj [url leaf $project]
  puts "$leafproj: "
  puts "$AllProjectsMap($leafproj)<br>"
}
```

The `$AllProjectsMap` array is keyed by ProjectName. The `$ProjectName` global variable stores the URL of the project, so the `url leaf` command is required to access just the leaf project name.

The substitution is included in the `EditTimecard.html` template as follows:

```
<TR>
  <!-- SYNC std_header "List Project Attributes" -->
  <TD valign=top>
    <!-- SYNC listattributes -->
  </TD>
</TR>
```

The resulting panel displays:



List Project Attributes	Triton: Owner karen Configs {Rel1.0 {Owner karen Description {release 1.0} FetchPreferences Rel1.0} Rel1.1 {Owner karen Description {1.1 release} FetchPreferences Rel1.1 Users {jblaine karen}}}

AllUsersList

The `AllUsersList` global variable provides a list of all users registered on the SyncServer, including their name attributes.

Type

The `AllUsersList` global variable is a list, sorted by user ID.

Format

`AllUsersList` has the following format:

```
{<user1> {Name {<User1 Name>}} <user2> {Name {<User2 Name>}}
<user3> {Name {<User3 Name>}} ...}
```

`<user>`	The user's ID. Required.
`Name {<User Name>}`	The `Name` identifier is followed by the user's full name contained in a list.

Example

The SyncServer has six users registered and stored in `AllUsersList`:

```
{alex {Name {Alessandro Rotolo}} anabel {Name {Anabel Blythe}}
cdent {Name {Charles Dent}} jboswell {Name {Jean Boswell}} josef
{Name Josef Schmidt} karen {Name {Karen Green}}}
```

AllUsersMap

The `AllUsersMap` global variable provides an array containing all users registered on the SyncServer, including their name attributes.

Type

The `AllUsersMap` global variable is an array, indexed by user ID.

Format

The format is identical to the `AllUsersList` variable, with the exception that the user IDs are used as the array indices. An array index value contains the user's full name. See `AllUsersList` for the syntax of the users list.

Example

The following substitution, defined in a custom panel initialization file, `share/panels/NoteDetail/EditTimecard.ini`, prints the complete name of the project owner by indexing the `AllUsersMap` variable:

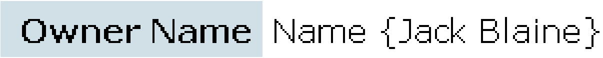
```
substitution ownername {} {
  global field_values
  global AllUsersMap

  set user $field_values(Owner)
  puts "$AllUsersMap($user)<BR>"
}
```

The substitution acquires the user's login name by from `$field_values` global variable. The `$AllUsersMap` array accepts the user's login name and returns the user's complete name.

The substitution is included in the `EditTimecard.html` template as follows:

```
<TR>
  <!-- SYNC std_header "Owner Name" -->
  <TD align=center>
    <!-- SYNC ownername -->
  </TD>
</TR>
```

The resulting panel displays:



Owner Name Name {Jack Blaine}

classes

The classes global variable is an array containing the class corresponding to each field (property) on a note. A field's class determines how it will be rendered and formatted in the note panel. A class is similar to the field's type, but in some cases the class is a derivative of a base type. For example, CC list is a configurable field which is a derivative of a String type.

Type

The classes global variable is an array, indexed by field name.

Format

To determine a field's class, access the classes array as follows:

```
$classes(<fieldname>)
```

Example

For diagnostics, the following code is included in a panel initialization file, Add<notetype>.ini:

```
puts "$classes(ShowStop)<BR>"
puts "$classes(Author)<BR>"
puts "$classes(Body)<BR>"
puts "$classes(DateCreate)<BR>"
puts "$classes(Id)<BR>"
```

The panel displays the following results:

```
boolean
userlist
transcript
date
number
```

defvals

The defvals global variable is an array containing the default value for each field (property) on a note. You define or change the default value of a field using the ProjectSync Note Type Manager.

Type

The defvals global variable is an array, indexed by field name.

Format

To determine a field's default value, access the `defvals` array as follows:

```
$defvals(<fieldname>)
```

Example

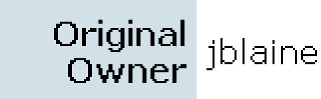
The following code in a panel initialization driver, `Edit<notetype>.ini`, defines a substitution that displays the Owner field's default value.

```
substitution ownerdef {} {
  global defvals
  puts "$defvals(Owner)"
}
```

The substitution is invoked in the corresponding `Edit<notetype>.html` file:

```
<TR>
  <!-- SYNC std_header "Original Owner:" -->
  <TD>
    <!-- SYNC ownerdef -->
  </TD>
</TR>
```

The HTML code generates this field containing the default value for the Owner field:



Original Owner jblaine

DisplayMode

The `DisplayMode` global variable indicates whether the panel mode is in Add, Edit, or View mode.

Type

The `DisplayMode` global variable is a string.

Format

The `DisplayMode` global variable contains the string `Add`, `View`, or `Edit` depending on the panel mode. `DisplayMode` is `View` by default.

Example

The following code checks which display mode is set and creates a customized substitution for Edit or View mode. You can include this code in separate file and source the file from the Edit<NoteType>.ini file and the View<NoteType>.ini file.

```
if {$DisplayMode == "Edit"} {
  substitution myfield {} {
    puts "Use the Edit mode to update an existing note."
  }
}

if {$DisplayMode == "View"} {
  substitution myfield {} {
    puts "Use the View mode to view an existing note \
     in read-only mode."
  }
}
```

field_values

The field_values global variable is an array containing the current value for each field (property) on a note.

Type

The field_values global variable is an array, indexed by field name.

Format

To determine a field's value, access the field_values array as follows:

```
$field_values(<fieldname>)
```

Example

The following code is included in a panel initialization file, Edit<notetype>.ini:

```
puts "$field_values(ShowStop)<BR>"
puts "$field_values(Author)<BR>"
```

The panel displays the following results:

```
False
karen
```

fields

The `fields` global variable is a list containing all fields (properties) of a note in display order.

Type

The `fields` global variable is a list.

Format

To list the fields of a note type in display order:

```
$fields
```

Example

The following code is included in a panel initialization file, `Edit<notetype>.ini`:

```
puts "$fields<BR>"
```

The panel prints a list of the fields of the note type:

```
Id DateCreate Author Title Subject WhenReslvd Actions Priority
Body moretext fileattach ShowStop External State Resolution
Owner myfield
```

NoteId

The `NoteId` global variable contains the ID number of the note being viewed or edited. The `NoteId` variable is set to 0 in Add mode.

Type

The `NoteId` global variable is an integer.

Format

To display the ID number:

```
$NoteId
```

Example

The following code is included in a panel initialization file, `EditYellowSticky.ini`:

```
puts "Note Id: $NoteId<BR>"
```

The panel prints the values of the specified global variable:

```
Note Id: 3
```

Module Name

The `ModuleName` global variable contains the current value of the project set for a note, if it is a module.

Type

The `ModuleName` global variable is a string.

Format

The `ModuleName` global variable contains the module specified for the note in the format:

```
sync:///Modules/<module>
```

If Note has no Module attachment this variable is empty (“”)

Example

The following code in a panel initialization driver, `Edit<notetype>.ini`, checks for a project value.

```
if {$ModuleName == "sync:///Modules/Thunder"} {
  puts "Module $ModuleName is EOL'ed.<BR>"
} else {
  puts "Module: $ModuleName<BR>"
}
```

For project Thunder, the following error message displays in the Edit panel:

```
Module sync:///Modules/Thunder is EOL'ed.
```

For projects other than Thunder, the project displays in the Edit panel:

```
Module: sync:///Modules/Asic
```

Module Tag

The ModuleTag global variable contains the current value of the configuration set for a note.

Type

The ModuleTag global variable is a string.

Format

The ModuleTag global variable contains a valid configuration for the module specified for the note in the format:

```
sync:///Modules/<module>;<selector>
```

For example:

```
sync:///Modules/Asic;B1:Gold
```

represents the Gold version of the B1 branch.

If the Module was attached to the Note without any “Tag” selection this variable is empty (“”)

Example

The following code in a panel initialization driver, Edit<notetype>.ini, checks for a configuration value.

```
if {$ModuleTag == ""} {
  puts "Please enter a configuration.<BR>"
} else {
  puts "$ModuleTag<BR>"
}
```

If a Module configuration value is set, the configuration value displays in the Edit panel for the note type:

```
sync:///Modules/Asic;Trunk:Gold
```

NoteSystem

The NoteSystem global variable contains the note system of the note being displayed. Currently, the only supported note system is SyncNotes.

Type

The `NoteSystem` global variable is a string.

Format

The `NoteSystem` variable represents the NoteSystem portion of a note's URL; for example, the portion of the URL below shown in bold:

sync:///Note/**SyncNotes**/YellowSticky/3

To display the note system:

```
$NoteSystem
```

Example

The following code is included in a panel initialization file, `EditYellowSticky.ini`:

```
puts "Note System: $NoteSystem<BR>"
```

The panel prints the value of the specified global variable:

```
Note System: SyncNotes
```

NoteType

The `NoteType` global variable contains the note type of the note being displayed.

Type

The `NoteType` global variable is a string containing one of the valid note types on the SyncServer.

Format

To display the note type of a note:

```
$NoteType
```

Access Control

When used as a note panel argument, access controls are obeyed. For example, if you invoke a Full Text Search via the URL http://{host}:{port}/scripts/isynch.dll?panel=TextSearch&NoteType=SyncDefect , if the user does not have permission to view notes of type SyncDefect, no matches will result. Access controls are also applied on a per note basis. In the SyncDefect search example, if the end user has permission to access SyncDefect notes in general, but not a particular SyncDefect note Id, that note Id will not be among the displayed search results, even if it matches the search criteria.

Example

The following code is included in a panel initialization file, `EditYellowSticky.ini`:

```
puts "Note Type: $NoteType<BR>"
```

The panel prints the values of the specified global variable:

```
Note Type: YellowSticky
```

NoteURL

The `NoteURL` global variable contains the full URL of the note being displayed.

Type

The `NoteURL` global variable is a string.

Format

The format of the note URL follows:

```
sync:///Note/<NoteSystem>/<NoteType>/<NoteId>
```

To display the note URL:

```
$NoteURL
```

Example

The following code is included in a panel initialization file, `EditYellowSticky.ini`:

```
puts "Note URL: $NoteURL<BR>"
```

The panel prints the values of the specified global variable:

```
Note URL: sync:///Note/SyncNotes/YellowSticky/3
```

other_attachments

The other_attachments global variable lists the attachments for the note, excluding the project and configuration attachments. The other_attachments global variable applies to Edit and View mode only.

Type

The other_attachments global variable is a list.

Format

The list of attachments contains the URL of each attachment.

To display the list of attachments:

```
$other_attachments
```

Example

The following code is included in a panel initialization file, EditSyncDefect.ini:

```
puts "Other attachments: $other_attachments<BR>"
```

The panel prints the list of attachments:

```
Other attachments: {sync:///Projects/ASIC1/x.v;1.1}
{sync:///Projects/Asic1/x.v;}
{sync:///Projects/Asic1/Add/shift.v;}
```

ProjectConfig

The ProjectConfig global variable contains the current value of the configuration set for a note.

Type

The ProjectConfig global variable is a string.

Format

The ProjectConfig global variable contains a valid configuration for the project specified for the note in the format:

```
sync:///Projects/<project>@<config>
```

Example

The following code in a panel initialization driver, `Edit<notetype>.ini`, checks for a configuration value.

```
if {$ProjectConfig == ""} {
  puts "Please enter a configuration.<BR>"
} else {
  puts "$ProjectConfig<BR>"
}
```

If a Project configuration value is set, the configuration value displays in the Edit panel for the note type:

```
sync:///Projects/Asic@Rel2
```

ProjectName

The `ProjectName` global variable contains the current value of the project set for a note.

Type

The `ProjectName` global variable is a string.

Format

The `ProjectName` global variable contains the project specified for the note in the format:

```
sync:///Projects/<project>
```

Example

The following code in a panel initialization driver, `Edit<notetype>.ini`, checks for a project value.

```
if {$ProjectName == "sync:///Projects/Thunder"} {
  puts "Project $ProjectName is EOL'ed.<BR>"
} else {
  puts "Project: $ProjectName<BR>"
}
```

For project Thunder, the following error message displays in the Edit panel:

```
Project sync:///Projects/Thunder is EOL'ed.
```

For projects other than Thunder, the project displays in the Edit panel:

```
Project: sync:///Projects/Asic
```

ProjectRelease

The `ProjectRelease` global variable contains the current value of the configuration set for a note.

Type

The `ProjectRelease` global variable is a string.

Format

The `ProjectRelease` global variable contains a valid release for the project specified for the note in the format:

```
sync:///Projects/<project>@<release>
```

Example

The following code in a panel initialization driver, `Edit<notetype>.ini`, checks for a configuration value.

```
if {$ProjectRelease == ""} {
  puts "Please enter a release.<BR>"
} else {
  puts "$ProjectRelease<BR>"
}
```

If a Project release value is set, the release value displays in the Edit panel for the note type:

```
sync:///Projects/Asic@Rel2
```

prompts

The `prompts` global variable is an array containing the prompt values for all fields (properties) on a note. You specify prompts for the fields when you create them using the ProjectSync Note Type Manager.

Type

The `prompts` global variable is an array, indexed by field name.

Format

To access a field's prompt:

```
$prompts(<fieldname>)
```

Example

The following code is included in a panel initialization file, `Edit<notetype>.ini`:

```
puts "$prompts(ShowStop)<BR>"
puts "$prompts(Author)<BR>"
```

The panel displays the following results:

```
ShowStopper
Author
```

ptypes

The `ptypes` global variable is an array containing the property types for all fields (properties) on a note.

Type

The `ptypes` global variable is an array, indexed by field name.

Format

To access a field's property type:

```
$ptypes(<fieldname>)
```

Example

```
The following code is included in a panel initialization file,
Edit<notetype>.ini:

puts "Field ShowStop: $ptypes(ShowStop)<BR>"
puts "Field Author: $ptypes(Author)<BR>"
puts "Field Body: $ptypes(Body)<BR>"
puts "Field DateCreate: $ptypes(DateCreate)<BR>"
puts "Field Id: $ptypes(Id)<BR>"
```

The panel prints a list of the fields and property types of the note type:

```
Field ShowStop: Boolean
Field Author: String80
Field Body: String
Field DateCreate: Date
Field Id: Integer
```

reqfields

The `reqfields` global variable is an array indicating which fields (properties) are required on a note.

Type

The `reqfields` global variable is an array, indexed by field name.

Format

To determine whether a field is required:

```
$reqfields(<fieldname>)
```

If 1 is returned, the field is required. If 0 is returned, the field is optional.

Example

The following code is included in a panel initialization file, `Add<notetype>.ini`:

```
if {$field_values(Title) == "" && $reqfields(Title) == 1} {
  puts "Please enter a title.<BR>"
}
```

If the Title field is empty, the following message displays in the Add panel:

```
Please enter a title.
```

SYNC_ClientInfo

The `SYNC_ClientInfo` global variable is an array that provides information about the user and client accessing the panel.

Type

The `SYNC_ClientInfo` global variable is an array.

Format

To access the `SYNC_ClientInfo` information:

```
  $SYNC_ClientInfo(parameter)
```

where parameter is one of the following:

AgentName	The name of the browser that sent the request, as reported by the browser. This encodes information such as browser name, version, and OS. The format differs depending on the browser.
Locale	Time zone information, passed in as the number of minutes to be subtracted from GMT. For example, five hours after GMT would be represented as 300 (5 times 60).
IPAddress	The IP address of the client.
UserName	The username of the user. This information is also available via the SYNC_User environment variable. Use SYNC_User if you only need user name information.

Example

The following code is included in a panel initialization file, Edit<notetype>.ini:

```
puts "Agent: $SYNC_ClientInfo(AgentName)<BR>"
puts "Locale: $SYNC_ClientInfo(Locale)<BR>"
puts "IPAddress: $SYNC_ClientInfo(IPAddress)<BR>"
puts "UserName: $SYNC_ClientInfo(UserName)<BR>"
```

The panel displays the following results:

```
Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Locale: 240
IPAddress: 64.192.76.101
UserName: karen
```

SYNC_Parm

The SYNC_Parm global variable is an array that contains the parameters passed to a panel with their associated values. **Note**: Instead of directly accessing this array, consider using the cgi_arg utility; the values cgi_arg returns are all fully decoded and ready for use.   See the the cgi_arg utility topic for details.

Type

The SYNC_Parm global variable is an array, indexed by parameter.

Format

To access the SYNC_Parm information:

```
  $SYNC_Parm(parameter)
```

where `parameter` is one of the parameters passed into the panel. For example, if a panel is executed via a panel URL, you can access the parameters as shown below:

```
http://<host>:<port>/scripts/isynch.dll?panel=TclScript
&file=filename&name=Joe&age=30&weight=160
```

then in order to access the name, age, and weight, you would use `$SYNC_Parm(name)`, `$SYNC_Parm(age)`, and `$SYNC_Parm(weight)`.

The array values for each argument are fully decoded and ready for use.

Note: `SYNC_Parm` does not support multi-valued arguments; to extract values of multi-valued arguments, use the `cgi_arg` utility.

Example

The following code is included in a panel initialization file, `Edit<notetype>.ini`:

```
puts "Note Type: $SYNC_Parm(NoteType)<BR>"
puts "Display Mode: $SYNC_Parm(DisplayMode)<BR>"
puts "Note ID: $SYNC_Parm(NoteId)<BR>"
```

The panel displays the following results:

```
Note Type: YellowSticky
Display Mode: Edit
Note ID: 2
```

See Also

Postprocessing in Panel Initialization Files

SYNC_Query

The `SYNC_Query` global variable contains the unfiltered query string supplied to the note panel, with the exception of the `panel=` portion of the string. The query values are HTML-encoded and must be decoded. To access the decoded parameters and values of the query, access the `SYNC_Parm` variable. **Note**: Instead of directly accessing these variables, consider using the `cgi_arg` Tcl utility.

Type

The `SYNC_Query` global variable is a string.

Format

To access the query string :

```
  $SYNC_Query
```

Example

The following code is included in a panel initialization file, `Edit<notetype>.ini`:

```
puts "SYNC_Query: $SYNC_Query<BR>"
```

The panel displays the following results:

```
SYNC_Query: NoteSystem=SyncNotes&DisplayMode=Edit&
NoteType=YellowSticky&NoteId=2
```

SYNC_User

The `SYNC_User` global variable contains the username of the user accessing the ProjectSync panel.

Type

The `SYNC_User` global variable is a string.

Format

To access the `SYNC_User` information:

```
  $SYNC_User
```

Example

The following code is included in a panel initialization file, `Edit<notetype>.ini`:

```
puts "SYNC_User: $SYNC_User<BR>"
```

The panel displays the following results:

```
SYNC_User: karen
```

Note Panel Arguments

Note Panel Arguments

Note panel arguments are the arguments supported for the Add, Edit, and View modes of panels. You can pass arguments to panels within the panel URL as described in URLs for Loading Panels.

In the following example, `panel` and `NoteType` are panel arguments:

```
http://myhost:2647/scripts/isynch.dll?panel=NoteDetail&NoteType
=
Note&NoteId=1
```

You can access these panel arguments in your `.ini` panel initialization drivers using the `cgi_arg` utility or the `SYNC_Parm` global variable.

The following note panel arguments are supported:

DisplayMode	Indicates whether the panel mode is in Add, Edit, or View mode. Optional. See Note Panel Global Variables: DisplayMode for more information.

LinkedURLs	Lets you supply initial object attachments for a note. These object URLs appear in the Attach To field of the note in Add mode. The `LinkedURLs` panel argument is optional, available only in the Add mode of note panels. `LinkedURLs` is a list of DesignSync object URLs, each separated by a new line character. The default value for `LinkedURLs` is `""` (no initial attachments).
	For example, you can seed the Attach To field by invoking the Add mode of the panel using a panel URL with the `LinkedURLs` argument, for example,
	`http://myhost:2647/scripts/isynch.dll?`
	`panel=NoteAdd&NoteType=TimeCard&LinkedURLs="sync://"`

NoteId	The ID number of the note being viewed or edited. Optional. See Note Panel Global Variables: NoteId for more information.
NoteSystem	The note system of the note being displayed; currently, the only supported note system is `SyncNotes`. See Note Panel Global Variables: NoteSystem for more information.
NoteType	The note type of the note being displayed. Optional. See Note Panel Global Variables: NoteType for more information.
NoteURL	The full URL of the note being displayed. Optional. See Note Panel Global Variables: NoteURL for more information.

panel	The name of the panel to execute and display. Required. For adding notes and displaying existing notes, the valid values are “NoteAdd” and “NoteDetail” respectively.
ProjectName	Initial Project attachment for the note. Optional. Specify the name of an existing top-level project. Defaults to ‶″ (no initial project attachment). Available in the NoteAdd panel only.
ProjectConfig	Initial Project Configuration attachment for the note. Optional. Specify the name of an existing configuration for the project attachment. Requires that an initial value for the ProjectName argument be supplied. Defaults to ‶″ (no initial configuration attachment). Available in the NoteAdd panel only.
<property name>	Lets you initialize any arbitrary property name to a value. Optional. Overrides any default value for a specific property on the note being added. The argument name must be a valid property name for the note type being added and the value must be a valid value for the property type. Available in the NoteAdd panel only.

Tcl Utilities

call_substitution

```
call_substitution <substitution> <arguments>
```

Description

Invoke the `call_substitution` utility to include a substitution within a Tcl panel script or within a custom substitution. The substitutions generally use the Tcl `puts` command to generate output, so you don't need to wrap the `call_substitution` call in a `puts` call.

Use

Use the `call_substitution` utility in Tcl panel scripts or within substitution definitions in `.ini` initialization drivers. You can use `call_substitution` with note panels, as well as custom panels. Use of the `call_substitution` utility is optional.

Arguments

`<substitution>`	Specify the name of a substitution, either a predefined substitution or one you have defined in a `.ini` panel initialization file.
`<arguments>`	Specify the arguments required by the substitution.

Globals Referenced

None.

Returns

Returns the output defined in the substitution.

Example

```
call_substitution field Title
```

This invocation in a Tcl panel script has the same effect as including the following substitution in a `.ini` panel initialization file:

```
<!-- SYNC field Title -->
```

cgi_arg

```
cgi_arg [-multi] <argname> [<defval>] [<varname>]
```

Description

The `cgi_arg` utility extracts the value of a panel argument for use in either a substitution or procedure. The `cgi_arg` utility lets you access panel parameters without having to access the `SYNC_Parm` array directly. The values `cgi_arg` returns are all fully decoded and ready for use.

Use

Use the `cgi_arg` utility in Tcl panel scripts or `.ini` initialization drivers. You can use `cgi_arg` with note panels, as well as custom panels. Use of the `cgi_arg` utility is optional.

Arguments

`-multi` Use the `-multi` option to extract an argument containing multiple elements. If `-multi` is supplied, `cgi_arg` returns a list, even if there is only one value. If a default value (`defval`) argument is specified and needs to be used, it is returned in a list.

`<argname>` Specifies the name of the argument whose value is to be extracted from the `SYNC_Parm` array. Required.

`<defval>` Specifies a default value to be returned if the argument specified (`argname`) is not present. Defaults to “”. Optional.

`<varname>` Lets you specify a new variable name to contain the value of `argname`. Defaults to the value of `argname`. Optional. To specify the `varname` argument, you must also specify the `defval` argument.

Globals Referenced

SYNC_Parm

Returns

No explicit return value. The specified panel argument value is returned to a variable whose name is passed by reference. The value returned is fully decoded HTML code.

Example

The following URL invokes a panel named StatusResult with parameters `name`, `month`, `projects`, and `report`:

```
http://myhost:2647/scripts/isynch.dll?panel=StatusResult
&name=Marvin Maxwell&month=October&projects=KI64&
projects=JI87&report=Completed project gh78
```

(To see the Status script that generates a URL to invoke the StatusResult panel, see Implementing Panels Using Tcl Panels Scripts.)

The `SYNC_Query` string generated by the URL follows:

```
name=Marvin%20Maxwell&month=October&projects=KI64
&projects=JI87&report=Completed%20project%20gh78
```

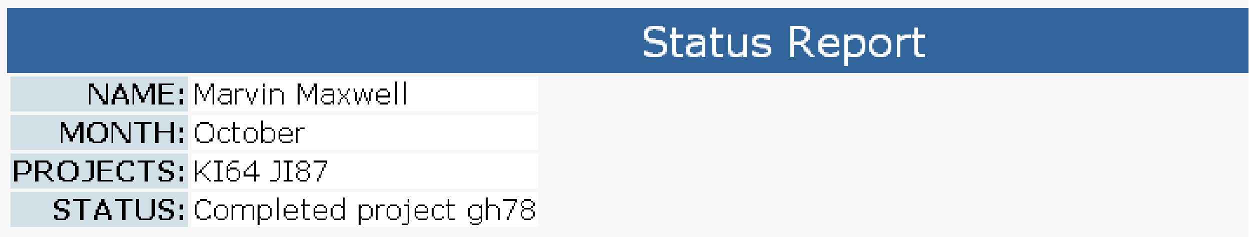
Notice that the `projects` parameter is a multi-valued parameter. Each separate project value is specified with its own instance of the `projects` parameter.

The following is the `StatusResult.tcl` script that displays these parameter values:

```
cgi_arg name
cgi_arg month
cgi_arg -multi projects
cgi_arg report

puts {
  <html><head><title>StatusResult</title></head>
}

call_substitution style

puts {
  <body CLASS=StatusResult>
  <table align="center" width="100%" cellspacing=0 cellpadding=4
border=0>
    <tr><th class="PAGETITLE">Status Report</th></tr>
  </table>
  <table class=LEFTHEADERTBL>
}

puts "<tr><th align=left>NAME:</th><td>$name</td></tr>"
puts "<tr><th align=left>MONTH:</th><td>$month</td></tr>"
puts "<tr><th align=left> PROJECTS:</th><td>$projects</td></tr>"
puts "<tr><th align=left>STATUS:</th><td>$report</td></tr>"

puts {
  </table></body></html>
}
```

Notice that the `projects` parameter requires the `-multi` switch to pass multiple values.

Following is the resulting panel:



Status Report

NAME:	Marvin Maxwell
MONTH:	October
PROJECTS:	KI64 JI87
STATUS:	Completed project gh78

encodeUrl

```
encodeUrl <string>
```

Description

The `encodeUrl` function encodes a URL, converting non-alphanumeric characters to the format `%XX`, where `XX` is the ASCII code of the character in hexadecimal. Spaces become plus signs. See URLs for Loading Panels for more information.

Use

Use the `encodeUrl` utility in server-side Tcl scripts including panel scripts and `.ini` initialization drivers. Use of the `encodeUrl` utility is optional.

Arguments

`<string>`	String to be encoded.

Globals Referenced

None.

Returns

Encoded string to be included in a URL.

Example

The following lines are included in a server-side script:

```
set encodedUrl [encodeUrl "name=Roger Smith&month=July"]
puts "$encodedUrl <br>"
```

The resulting string follows:

```
name%3dRoger+Smith%26month%3dJuly
```

You can then include the string to invoke a panel URL, for example:

```
http://myhost:2647/scripts/isynch.dll?panel=StatusResult
&name%3dRoger+Smith%26month%3dJuly
```

htmlResult

```
htmlResult [-error <error_message> |
-warning <warning_message>] [<explanation>] [<args>]
```

Description

You invoke the `htmlResult` utility within Tcl scripts or panel initialization files to generate a standard results message.

Use

Use the `htmlResult` utility within Tcl panels scripts and `.ini` panel initialization driver files such as `EditNote.ini`. Use of `htmlResult` is optional.

Note: You can invoke the `htmlResult` utility within a Process initialization driver (for example, `ProcessNote.ini`); however, the `htmlResult` utility is not supported within the `Process_Note_PreSchema` and `Process_Note_PostSchema` procs that you can define in a Process initialization driver.

Arguments

`-error|-warning` Indicates the general condition being reported. Optional. These switches control the appearance of the generated result page. If `-error` is specified, the default title reads “Operation Failed” and the explanatory table details display in shades of red. The `-warning` argument causes the default title to read "Operation Successful” and the explanatory table details display in shades of yellow .

`-title` Controls the content of the single line of explanatory text. Optional. The default value is “Operation Successful” (unless the `-error` switch is used). The `-title` switch takes precedence over the titles specified with the `-error` and `-warning` switches.

`<explanation>` Controls the text of the second column of the explanatory table details. Required if `-error` or `-warning` are not specified. The first column is always labeled “Action”.

`<args>` Lets you add rows to the explanatory table. Optional. Additional arguments must be given in pairs: the first being the text of the left column of a table row in the explanatory table, the second being the right column text for the same row. Each pair of arguments defines a new row of information for the explanatory table:

```
<Row1Column1> <Row1Column2> <Row2Column1>
<Row2Column2> ...
```

If neither the `-error` or `-warning` switches are specified, you can control the color of a specific row by appending a single or double exclamation point to the first argument of that row:

- A single exclamation point (!) creates a yellow (warning) row, for example, `"Important!"`
- Double exclamation points (!!) creates a row in red (error) color, for example, `"Critical!!"`

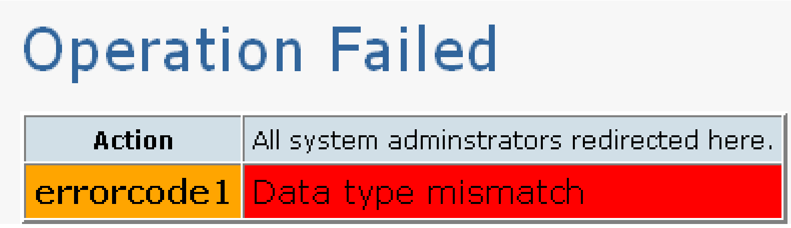
Globals Referenced

None.

Examples

```
cgi_arg position

if {$position == "SysAdmin"} {

  htmlResult "All system adminstrators redirected here." \
  -error "errorcode1" "Data type mismatch"

  select_template SysAdminFeedback

}
```

The `htmlResult` invocation displays these results:



Operation Failed

Action	All system adminstrators redirected here.
errorcode1	Data type mismatch

select_template

`select_template <template_name>`

Description

The `select_template` utility switches control to a different HTML template and `.ini` initialization driver pair. The HTML template you specify can have its own `.ini` file that ProjectSync also processes. When ProjectSync processes the `select_template` expression, it looks for the required `<template_name>.html` file and for the optional `<template_name>.ini` file.

If the alternate panel is instead implemented as a Tcl script, use the `select_tcl_script` utility.

Use

Use the `select_template` utility in Tcl panel scripts or `.ini` initialization drivers. You can use `select_template` with note panels, as well as custom panels. Use of the `select_template` utility is optional.

Arguments

`<template_name>`	Specify the name of the HTML template and `.ini` initialization driver pair to be loaded (without the filename extensions). (Required) The `<template_name>.html` file must exist for the template to be switched. ProjectSync searches for this file in the current panel directory (the directory containing the Tcl script or `.ini` file that is invoking the `select_template` utility).

Globals Referenced

None.

Returns

No explicit return value. ProjectSync loads and executes the template and any `.ini` files associated with the `<template_name>` argument.

Example

The following code is contained in the `/share/panels/NoteAdd/AddFeedback.ini` file:

```
if {$position == "SysAdmin"} {
  select_template SysAdminFeedback
}
```

ProjectSync loads the `SysAdminFeedback.html` template, as well as the `SysAdminFeedback.ini` file, if it exists.

substitution

```
substitution <subst_name> <subst_arg_list> <subst_code>
```

Description

All panels can define their own set of substitutions. The `substitution` utility lets you create custom substitution tags with new behaviors from those of the predefined HTML substitution tags described in Note Panel Substitution Tags. If a substitution definition for a new tag is defined in a panel's `.ini` panel initialization file, you can then include that substitution tag in the corresponding HTML template file.

Use

Use the `substitution` utility to define custom substitutions in Tcl panel scripts or `.ini` initialization drivers. To invoke the substitution, either include the substitution tag in your panel HTML file (see What Are Substitution Tags?) or call the substitution from a Tcl panel script using the call_substitution command. You can use `substitution` with note panels, as well as custom panels. Use of the `substitution` utility is optional.

Arguments

`<subst_name>` Specifies the name of the substitution tag you are generating. Substitution tag names are case insensitive, like HTML tags. Required.

`<subst_arg_list>` Defines the parameters you must pass to the tag in the HTML template. Optional. Specify order-dependent parameters as you would for a Tcl `proc`:

```
{required1 required2 -option1 -option2 ...}
```

The optional arguments must follow the required arguments. You can also provide default values for parameters:

```
{{required1 ""} {required2 ""} {-option1 6}
{-option2 ""}}
```

`<subst_code>` Contains the Tcl code you want executed when ProjectSync encounters this tag in the HTML template. To implement the HTML constructs you want displayed in the note panel, you include puts statements in the subst_code parameter's Tcl code. Required.

Globals Referenced

None.

Returns

No explicit return value. As ProjectSync parses the custom substitution tag in an HTML panel template, it displays the HTML specified in the custom substitution.

Example

Example: A Substitution to Generate a Banner

The following example of a substitution subroutine generates a banner in a note panel if the substitution tag is included in the panel's HTML template:

```
substitution add-banner {} {
  puts "<img src=/synccontent/images/bar.gif>"
}
```

The substitution code above is included in the panel initialization (.ini) file.

In this example, synccontent is an alias for the <SYNC_DIR>/share/content directory. (You cannot specify absolute file system paths on the SyncServer.) See DesignSync Data Manager Administrator's Guide: SyncServer Aliases for more information on SyncServer aliases.

You include the substitution tag in the panel's HTML template as follows:

```
<HTML>
  ...
  ...
<!-- SYNC add-banner -->
Title <!-- SYNC field Title -->
  ...
</HTML>
```

Example: A Substitution with Parameters

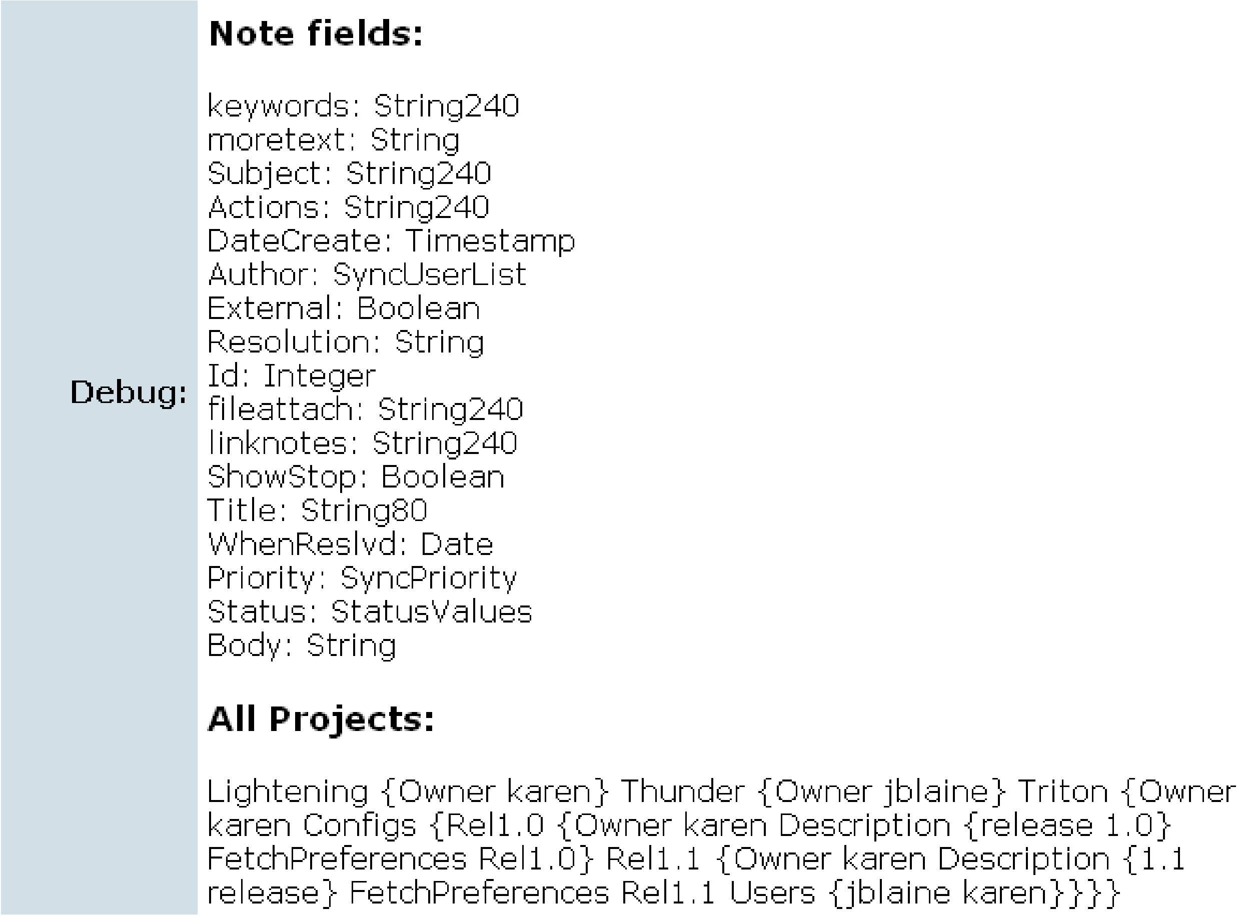
The following substitution contained in the note type's panel initialization (.ini) file dumps the values of fields and projects. The substitution illustrates the use of required and optional parameters. The `topanel` argument is required and the `notefields` and `allprojects` arguments are optional.

```
substitution dumpvars {topanel -notefields -allprojects} {
global AllProjectsList

  if {$topanel != 0} {
    if {$notefields != 0} {
      puts "<H3>Note fields:</H3>"
      set field_names \
          [note schema -ptypes types "TimeCard"]
      foreach field $field_names {
        puts "$field: $types($field)<BR>"
      }
    }
  }

  if {$allprojects != 0} {
    puts "<H3>All Projects:</H3>"
    puts "$AllProjectsList<BR>"
  }
}
```

You include the substitution tag in the panel's HTML template as follows:

```
<TR>
  <!-- SYNC std_header "Debug:" -->
    <TD align=center>
      <!-- SYNC dumpvars 1 -notefields -allprojects -->
    </TD>
</TR>
```

The results of the `dumpvars` substitution follow:



Debug:

Note fields:

keywords: String240
moretext: String
Subject: String240
Actions: String240
DateCreate: Timestamp
Author: SyncUserList
External: Boolean
Resolution: String
Id: Integer
fileattach: String240
linknotes: String240
ShowStop: Boolean
Title: String80
WhenReslvd: Date
Priority: SyncPriority
Status: StatusValues
Body: String

All Projects:

Lightening {Owner karen} Thunder {Owner jblaine} Triton {Owner karen Configs {Rel1.0 {Owner karen Description {release 1.0} FetchPreferences Rel1.0} Rel1.1 {Owner karen Description {1.1 release} FetchPreferences Rel1.1 Users {jblaine karen}}}}

Other Utilities

The following Tcl utilities might also be helpful in your customizations:

caught_error	Use the `caught_error` utility to signal abnormal termination due to an internal error. Specify a string as an argument to `caught_error`; the top-level template processor catches the error and displays the specified message string in a diagnostics page.
include	Use the `include` utility within your Tcl panel script or your `.ini` panel initialization file to include auxiliary source files. Specify the leaf name of the file to be included. ProjectSync searches for file in the most custom `/share/panels` directory, then in the most custom `/share/tcl` directory.
locate	Use the `locate` stcl command to search the DesignSync paths for a specified object, either a file or directory. You can find either the first occurrence of the object (the default) or all occurrences of the object.

quit	Use the `quit` utility to terminate any further processing of the current panel. No error message is displayed. Use `quit` for normal or abnormal termination, but you must display the results before calling `quit`. You must call `quit` to break out of `.ini` panel initialization file processing before the substitution processing phase.
select_tcl_script	To explicitly run a Tcl panel script, use `select_tcl_script`. Like `select_template`, you specify the name of the template: `select_tcl_script <template_name>` where `template_name` is the name of the Tcl file without the `.tcl` extension.
setDynamicUserListProps	Use the `setDynamicUserListProps` utility to control the list of SyncUserList properties that are automatically updated on a panel in Add or Edit mode as note attachments are made. Specify an optional list of fields -- these are the properties to be considered dynamic user list properties. The properties specified must be valid properties of the userlist class. The default is an empty list, indicating that there are no dynamic user list properties.
setFieldOrder	Use the `setFieldOrder` utility to control the order and the set of properties to be included on a note panel. As an argument to `setFieldOrder`, specify a Tcl list containing the properties to be used in the order you want. Any properties not in the list are discarded.
var_substitution	Use the `var_substitution` utility as a shorthand for outputting the value of a global Tcl variable using this syntax: `var_substitution <name> {var_name “”} {-nonempty} {-quoted}` The optional `var_name` argument lets you define a shorthand alias, to index an array, for example. The `-nonempty` option generates an HTML non-breakable space (&nbsp;) if the variable's value is empty (""). This option is handy for inserting values into tables without having to worry about empty strings The `-quoted` option escapes all left angle brackets in the output.

Case Study: Design Tool Process Survey

Design Tool Process Survey

The following case study describes an online survey implemented as a note panel. The Feedback survey is the type of questionnaire that might be published by an internal CAD group to query for tool and process improvements. This survey could be implemented as a panel from scratch or as a note panel. We chose to implement the survey as a note panel. In this way, we can use the Note Type Manager to create the new note type named **Feedback** and the basic user interface. Then we can modify the HTML and add `.ini` panel initialization scripts to further customize the panels.

Because we are implementing the survey as a note, users can complete a survey by selecting **Add Feedback** from the ProjectSync main menu. (When you create a note type, ProjectSync automatically creates an **Add <Note Type>** entry for the note type in the main menu.)

The case study walks through the design of the Feedback note type and the graphical interface panels that implement the survey.

Develop Paper Prototype of Survey

Before you begin designing a note type and graphical interface panels, it's a good idea to jot down a paper prototype of your design ideas. The following paper prototype shows the fields for the Designer survey, as well as those for the System Administrator survey.

CAE Process and Tools Survey Prototype

Designer Survey Prototype:

You could WIN a T-shirt and mug – just complete the survey below and click Submit. The 100th survey entry wins!

- Position: Design Lead, Designer, Test Engineer, Systems Administrator
- If you are a systems administrator, has your group upgraded to OS Version8.0? Yes/No
- If Design Lead, Designer, Test Engineer, the panel includes these fields:
- HDL Language Pulldown: Verilog/VHDL
- Other: Type-in field
- Size of Typical Module # gates
- Simulation Tool: Pulldown: VQuick/SimQuick/Simmer/NA
- This tool is effective. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree

- This tool is easy to use. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- Synthesis Tool: Pulldown: BuildIt/GateGo/NA
- This tool is effective. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- This tool is easy to use. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- Layout Tool: Pulldown: LayoutMaster/SmartPlacer/GatePlace
- This tool is effective. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- This tool is easy to use. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- Physical Verification Tool: Pulldown: Verify/RightDesign
- This tool is effective. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- This tool is easy to use. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- Configuration Management Tool Pulldown: DesignSync/ReVision
- This tool is effective. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- This tool is easy to use. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- Design Chain Publishing Tool Pulldown: Publisher Suite/PassItOn
- This tool is effective. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- This tool is easy to use. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- Easiest tool to use: Pulldown: VQuick/SimQuick/Simmer/BuildIt/GateGo/Verify/RightDesign/ReVision/Publisher Suite/DesignSync/PassItOn
- Most difficult tool to use: Pulldown: VQuick/SimQuick/Simmer/BuildIt/GateGo/Verify/Publisher Suite/RightDesign/ReVision/DesignSync/PassItOn
- Please specify the degree to which you agree with the following statements:
- Our tools are state-of-the-art design tools. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- I find the tool flow to be straightforward. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- The hand-off between design and layout is smooth and efficient. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- The tool flow and processes are well documented. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- I have a clear line of communication with other groups in the design chain. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- Please describe problems you have had with specific tools. Multi-line type-in field

- Please describe problems you have had at any point during design flow. Multi-line type-in field

System Administrator Survey Prototype:

If System Administrator, the following fields display:

- How many hours did it take to install the upgrade the OS:

Please answer the following questions:

- The upgrade was straightforward. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- Our tools are state-of-the-art design tools. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- I find the tool flow to be straightforward. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- The hand-off between design and layout is smooth. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- Iterations between design stages are efficient. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree
- I have a clear line of communication with other groups in the design chain. Radio button: Strongly agree/Somewhat agree/Agree/Somewhat disagree/Strongly disagree

Develop Property Types

If your note type requires custom property types, it's a good idea to create the property types before you create your note type. The Feedback note panels require custom pull-down fields, such as the HDL Language field. To create a custom property type, you bring up the Note Type Manager and select Manage Property Types. See ProjectSync User's Guide: Creating New Property Types.

For the Feedback note type, we create the following choice list property types:

Property Name	**Choice List Values**
HDLLanguage	Verilog, VHDL, Other, N/A
SimTools	Vquick, SimQuick, Simmer, Other, N/A
SynTools	BuildIt, GateGo, Other, N/A
LayoutTools	LayoutMaster, SmartPlace, GatePlace, Other, N/A
PhysVerTools	Verify, RightDesign, Other, N/A
DesMgmtTools	DesignSync, ReVision, Other, N/A
IPReuseTools	Publisher Suite, DesignChain, Other, N/A
AllTools	Vquick, SimQuick, Simmer, BuildIt, GateGo, LayoutMaster, SmartPlace, GatePlace, Verify, RightDesign, DesignSync,

	ReVision, Publisher Suite, DesignChain, Other, N/A
SurveyResponse	Strongly agree, Somewhat agree, Agree, Somewhat disagree, Strongly disagree
Position	String (**Note**: We will be changing this field to a custom JavaScript field in a subsequent section, Redirect to System Administration Panel.)

Create Feedback Note Type

The next step is to create the basic note type for the feedback survey using the Note Type Manager. See ProjectSync Help: Creating Custom Note Types.

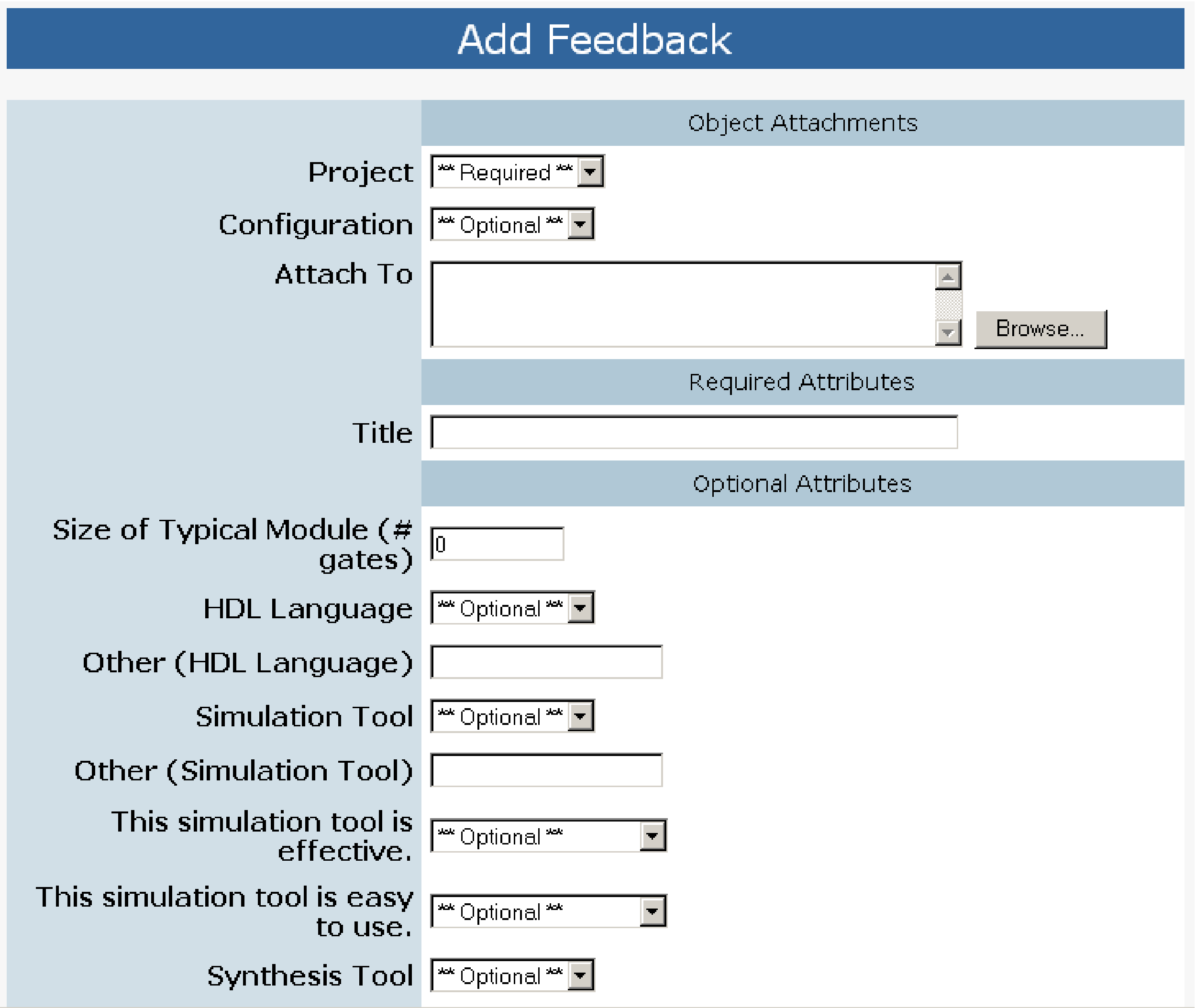
This first pass at creating the feedback user interface provides the survey questions only for the designers. A separate panel will be created later with questions directed specifically toward system administrators.

We add the following properties to the basic Feedback Note Type.

Field (Property)	**Field Type Property Type)**	**Prompt**
Position	String20	Position (**Note**: We will be changing this field to a custom JavaScript field in a subsequent section, Redirect to System Administration Panel.)
ModSize	Integer	Size of Typical Module (# gates)
HDLLang	HDLLanguage	HDL Language
HDLOther	String20	Other (HDL Language)
SimTool	SimTools	Simulation Tool
SimOther	String20	Other (Simulation Tool)
SimEffect	SurveyResponse	This simulation tool is effective.
SimEasy	SurveyResponse	This simulation tool is easy to use.
SynTool	SynTools	Synthesis Tool
SynOther	String20	Other (Synthesis Tool)
SynEffect	SurveyResponse	This synthesis tool is effective.
SynEasy	SurveyResponse	This synthesis tool is easy to use.
LayTool	LayTools	Layout Tool
LayOther	String20	Other (Layout Tool)
LayEffect	SurveyResponse	This layout tool is effective.
LayEasy	SurveyResponse	This layout tool is easy to use.
PVTool	PVTools	Physical Verification Tool
PVOther	String20	Other (Physical Verification Tool)
PVEffect	SurveyResponse	This physical verification tool is effective.

PVEasy	SurveyResponse	This physical verification tool is easy to use.
DMTool	DMTools	Design Management Tool
DMOther	String20	Other (Design Management Tool)
DMEffect	SurveyResponse	This design management tool is effective.
DMEasy	SurveyResponse	This design management tool is easy to use.
IPTool	IPTools	IP Reuse Tool
IPOther	String20	Other (IP Reuse Tool)
IPEffect	SurveyResponse	This IP tool is effective.
IPEasy	SurveyResponse	This IP tool is easy to use.
Easiest	AllTools	The easiest design tool I use is:
Difficult	AllTools	The most difficult design tool I use is:
StateArt	SurveyResponse	Our tools are state-of-the-art design tools.
ToolFlow	SurveyResponse	I find the tool flow to be straightforward.
HandOff	SurveyResponse	The hand-off between design and layout is smooth and efficient.
Documented	SurveyResponse	The tool flow and processes are well documented.
Commune	SurveyResponse	I have a clear line of communication with other groups in the design chain.
ToolProbs	String	Please describe problems you have had with specific tools.
FlowProbs	String	Please describe problems you have had at any point during design flow.

A portion of the Feedback note type is shown in Add mode below. The pull-down menus in the note type correspond to the property types created in the previous step.



Add Feedback

	Object Attachments
Project	** Required ** ▾
Configuration	** Optional ** ▾
Attach To	Browse...
	Required Attributes
Title	
	Optional Attributes
Size of Typical Module (# gates)	0
HDL Language	** Optional ** ▾
Other (HDL Language)	
Simulation Tool	** Optional ** ▾
Other (Simulation Tool)	
This simulation tool is effective.	** Optional ** ▾
This simulation tool is easy to use.	** Optional ** ▾
Synthesis Tool	** Optional ** ▾

Generate HTML Templates

Once you have used the Note Type Manager to create a basic user interface, you can generate the HTML templates if you want to customize the presentation of the panel. In our case, we'll be creating extra panels for the Feedback note type, so we'll use the generated HTML Feedback templates as a starting point for our custom panels. See Generating HTML Templates for the steps to generate templates. For the Feedback note type, we generate the following templates:

```
share/panels/NoteAdd/AddFeedback.html

share/panels/NoteDetail/EditFeedback.html

share/panels/NoteDetail/ViewFeedback.html
```

Modify the AddFeedback Panel

The AddFeedback panel displays Configuration, Attach To, Title, and Note Text fields which are not relevant because our note panel is a survey. We do not need to track these fields for the Feedback notes.

Remove unnecessary headings and fields:

We delete the configuration and attachment substitutions from the `AddFeedback.html` file, as well as the Title and Body fields. We delete the headers and prompts associated with these fields, as well.

Redirect to System Administration Panel

We will modify the Position field so that it determines if the user surveyed is a System Administrator. If the user is a System Administrator, ProjectSync will display a different survey panel. We will convert the Position field into a radio button containing job titles. The button corresponding to "System Administrator" will execute a simple JavaScript script that will invoke the URL of the alternate panel, SysAdminFeedback.

We create the new panel, named `SysAdminFeedback.html` in the `/share/panels/NoteAdd` directory. To add new fields, we modify the Feedback note type using the Note Type Manager. Because the SysAdminFeedback panel contains some of the same survey questions as the general AddFeedback panel, we make a copy of the `AddFeedback.html` panel and name it `SysAdminFeedback.html`. Then, we add the extra fields and remove unnecessary fields from `SysAdminFeedback.html`.

Next, we change the Position field substitution in the `AddFeedback.html` file sot that it redirects to the alternate System Administrator panel. The original code generated during the HTML Generation step is shown below, followed by the new code that implements the custom field.

Position Field Code Generated during HTML Generation

```
<TR>
  <TH align=right><!-- SYNC prompt Position --></TH>
  <TD>
    <!-- SYNC field Position -->
  </TD>
</TR>
```

New Position Field Code that Redirects to SysAdminFeedback Panel

The following code implements the Position radio button in the `AddFeedback.html` file:

```
<TR>
  <TH align=right><!-- SYNC prompt Position --></TH>
  <TD>
    <INPUT TYPE=RADIO NAME=Position VALUE="Designer">
      Designer
    <INPUT TYPE=RADIO NAME=Position VALUE="Lead Design">
      Lead Design <BR>
    <INPUT TYPE=RADIO NAME=Position VALUE="Test Engineer">
      Test Engineer
    <INPUT TYPE=RADIO NAME=Position VALUE="System Administrator"
    onClick='Position_Redirect()'> System Administrator
    <BR>
    <B>Note</B>: System Administrators will be redirected
    <BR>
    to a different feedback form.

    <SCRIPT>
      function Position_Redirect()
      {
  location='isynch.dll?panel=NoteAdd&NoteType=Feedback&position=
SysAdmin

';
      }
    </SCRIPT>
  </TD>
</TR>
```

The AddFeedback panel now displays as follows. (Not all fields are shown.)

The URL invoked by the `Position_Redirect` function invokes the AddFeedback panel, with a custom parameter named `Position` set to `SysAdmin`.

The radio button displays in the AddFeedback panel as follows:

In order for the System Administration radio button to redirect to the System Administration feedback panel, the `AddFeedback.ini` file uses the `select_template` command to switch from the AddFeedback panel to the SysAdminFeedback panel:

```
cgi_arg position
```

```
if {$position == "SysAdmin"} {
  select_template SysAdminFeedback
}
```

Now, when the user clicks on System Administration in the Position radio button, the System Administration Feedback panel displays. (Not all fields are shown.)

Getting Assistance

Using Help

ENOVIA Synchronicity DesignSync Data Manager Product Documentation provides information you need to use the product effectively. The Online Help is delivered through WebHelp® , an HTML-based format.

Note:

Use SyncAdmin to change your default Web browser, as specified during DesignSync product tools installation. See SyncAdmin Help for details.

To bring up the online help from the tool you are using, do one of the following:

Select **Help => Help Topics** from the tool you are using. The help system opens in your default browser. The Contents tab displays in the left pane and the corresponding help topic displays in the right pane.

- Click **Help** on forms. The help system opens to the topic that describes the form.
- Press the **F1** key. The help system opens to the topic that describes the current form or window you have open.

To bring up stand-alone Online Help, do one of the following:

- Enter the correct URL from your Web browser:

  ```
  http://<host>:<port>/syncinc/doc/<docname>/<docname.htm>
  ```

- where <host> and <port> are the SyncServer host and port information. Use this server-based invocation when you are not on the same local area network (LAN) as the DesignSync installation.

  For example:

  ```
  http://<host>:<port>/syncinc/doc/pscustom/pscustom.htm
  ```

  ```
  http://<host>:<port>/syncinc/doc/DesSync/dessync.htm
  ```

- Enter the following URL from your Web browser:

  ```
  file:///$SYNC_DIR/share/content/doc/<docname>/<docname.htm>
  ```

  For example:

```
file:///$SYNC_DIR/share/content/doc/pscustom/pscustom.htm

file:///$SYNC_DIR/share/content/doc/DesSync/dessync.htm
```

where $SYNC_DIR is the location of the DesignSync installation. Specify the value of SYNC_DIR, not the variable itself. Use this invocation when you are on the same LAN as the installation. This local invocation may be faster than the server-based invocation, does not tie up a server process, and can be used even when the SyncServer is unavailable.

When the Online Help is open, you can find information in several ways:

- Use the **Contents** tab to see the help topics organized hierarchically.
- Use the **Index** tab to access the keyword index.
- Use the **Search** tab to perform a full-text search.

Within each topic, there are the following navigation buttons:

- **Show** and **Hide**: Clicking these buttons toggles the display of the navigation (left) pane of WebHelp, which contains the Contents, Index, and Search tabs. Hiding the navigation pane gives more screen real estate to the displayed topic. Showing the navigation pane givens you access to the Contents, Index, and Search navigation tools.
- **<<** and **>>**: Clicking these buttons moves you to the previous or next topic in a series within the help system.

You can also use your browser navigation aids, such as the **Back** and **Forward** buttons, to navigate the help system.

Getting a Printable Version of Help

The *ProjectSync Advanced Customization Guide* is available in book format from the ENOVIA Documentation CD or the DSDocumentationPortal_Server installation available on the 3ds support website. The content of the book is identical to that of the help system. Use the book format when you want to print the documentation, otherwise the help format is recommended so you can take advantage of the extensive hyperlinks available in the DesignSync Help.

You must have Adobe® Acrobat® Reader™ Version 8 or later installed to view the documentation. You can download Acrobat Reader from the Adobe web site.

Contacting ENOVIA

For solutions to technical problems, please use the 3ds web-based support system:

http://media.3ds.com/support/

From the 3ds support website, you can access the Knowledge Base, General Issues, Closed Issues, New Product Features and Enhancements, and Q&A's. If you are not able to solve your problem using this information, you can submit a Service Request (SR) that will be answered by an ENOVIA Synchronicity Support Engineer.

If you are not a registered user of the 3ds support site, send email to ENOVIA Customer Support requesting an account for product support:

enovia.matrixone.help@3ds.com

Related Topics

Using Help

Index