

---

# TACTIC Setup

## Table of Contents

Introduction .....	3
Introduction .....	3
TACTIC Anatomy Lesson .....	4
Built-in STypes .....	7
TACTIC Sidebar .....	9
Projects .....	14
Create a New Project .....	14
Projects .....	16
Project Schema .....	16
Register sTypes .....	23
Connecting sTypes .....	27
Projects .....	29
Project Workflow Introduction .....	29
Workflow Editor .....	31
Pipeline Process Options .....	37
Interface Setup .....	39
Managing the Sidebar .....	39
Element Definition Widget .....	43
View Manager .....	50
Users Management .....	53
Insert a New User .....	53
Managing Access Rules .....	55
Project Automation .....	57
TACTIC Event System Introduction .....	57
Project Automation - Triggers .....	59
Project Automation - Notifications .....	64
Project Automation - File Naming .....	68
Expression Language .....	76
TACTIC Expression Language Introduction .....	76
Expression Method Reference .....	80
Expression Variable Reference .....	85
Import and Export .....	87
Importing CSV Data .....	87
Exporting CSV Data .....	91
Advanced Configuration .....	94
Modify Project Settings .....	94
Advanced Schema Configuration .....	96
Advanced Access Rule Configuration .....	97
Remove Projects .....	101
Advanced Automation .....	102
Advanced Notification Setup .....	102
Debugging TACTIC .....	110
Exception Log .....	110
Submitting a Support Ticket .....	112
TACTIC Widgets .....	113
TACTIC Widgets .....	113
View Widgets .....	114

Simple Table Element .....	114
Formatted Widget .....	116
Expression .....	119
Expression Value Element .....	122
Button .....	124
Link Element .....	126
Gantt .....	127
Hidden Row .....	131
Drop Item .....	133
Edit Widgets .....	136
Select .....	136
Text Input .....	138
Text Area .....	140
Calendar Input Widget .....	141
Common Widgets .....	143
Completion .....	143
Explorer Button .....	146
General Check-in Widget .....	148
Checkin History .....	158
Note .....	160
Note Sheet Widget .....	162
Preview .....	164
Task Edit .....	166
Task Schedule .....	168
Task Status Edit .....	170
Task Status History .....	172
Work Button .....	173
Work Hours List .....	182
Layout Widgets .....	183
View Panel .....	183
Custom Layout .....	185
Table Layout .....	191

# Introduction

## Introduction

**The role of Setting up TACTIC is to configure and maintain the structure of the TACTIC projects.**

These responsibilities may include:

- Creating Projects
- Defining Project Schema
- Defining Project Workflow
- Managing Users and Groups
- Configuring Group Access Rules
- Managing the Project Sidebar
- Automating Notifications and Processes using Triggers.
- Defining naming conventions for the File system

This section will help you to understand how to approach this setup and configure your system properly.

# TACTIC Anatomy Lesson

## What is TACTIC and how is it all put together?

TACTIC is a project-based system that can be configured to accommodate the custom requirements of many different project scenarios.

Any project or asset management scenario can have a large number of items to manage. These items can be people, files, tasks or information and the management of these items are often a hurdle. The primary goal of TACTIC is to assist in generating placeholders, controlling work-flow and managing these items.

## The TACTIC Server

The Tactic server is a system which runs the TACTIC application. This server is often housed in your facility as a website which runs on your private network, or can be opened up to the world wide web like any other website.

For Tactic to run there are 4 main services:



<b>Database Server</b>	<p><b>This is where Tactic stores all of the meta data</b></p> <p>The "database is the base location for all of your data. This data is all information often tracked in spreadsheets, emails, sticky notes, whiteboards etc. A database provides a powerful central location for this information which helps keep everything in one place.</p>
<b>File server</b>	<p><b>This is were TACTIC stores the files.</b></p> <p>Files in TACTIC are managed externally on a file system. For example, most businesses have a file server that users save to. The file server typically has a root 'assets' directory where TACTIC handles the directory and filenames for the entire file system.</p>
<b>TACTIC Application</b>	<p><b>This is the central hub for all processing.</b></p> <p>All processes and interactions are managed through TACTIC transaction system. The TACTIC Application sends/retrieves information from the database, and files from the File Server. TACTIC's web interface can be delivered in multiple configurations based on the needs of the end user.</p>
<b>Web Server</b>	<p><b>The Web Server delivers the interface to the end user</b></p> <p>The Web Server delivers the TACTIC Interface to the end user's web browser. The Web Server is a web portal to the TACTIC Application.</p>

## What is a TACTIC Project?

A TACTIC project stores all inserted information and configuration. In the back-end, each project is a complete "database".

**There are 2 major components to a TACTIC Project database; Setup (configuration) and Meta Data**

**Project configuration** - Each TACTIC project can be unique based on the desired end-user experience. TACTIC is extremely configurable which make various end-user work-flows possible.

**Project Information (Meta Data)** - Once you have a project setup and configured, it stores all that data in the project. Because all data is centralized in a database, it makes real-time updates and collaboration on project tasks possible.

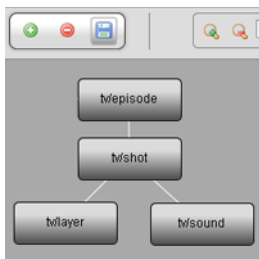
Tactic can house multiple projects at the same time with each being a separate project configuration. Within a project, there can be a hierarchy of different types of objects in your project design.

## Project Structure

There are 2 major components to setting up the base structure of a TACTIC Project:

- The **Project Schema** which represents **what you manage and produce, and how these objects relate to each other**.
- The **Project Work-flow** which represents **processes and work-flows these objects travel through during their life-cycle**

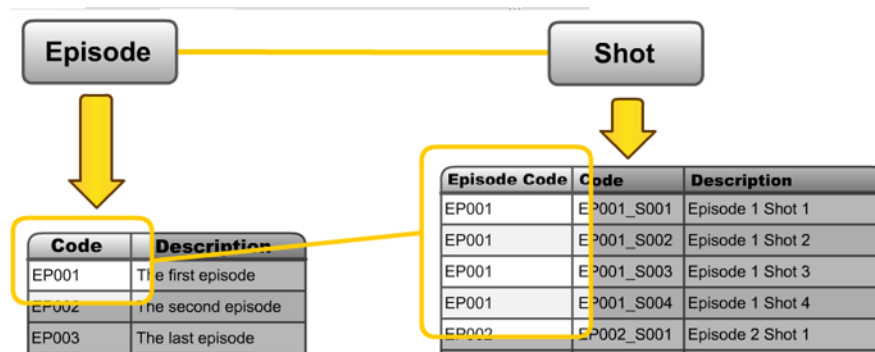
## Project Schema



The project Schema is the central hub for traversing a project, and is the most important aspect of the project setup. The Schema node (sType), is a type of object you manage. i.e.. episodes, shots, sound.

For example, a television series may have multiple episodes, and each episode may have multiple shots. By relating Episodes to Shots, this will display all shots in a particular episode in one TACTIC view.

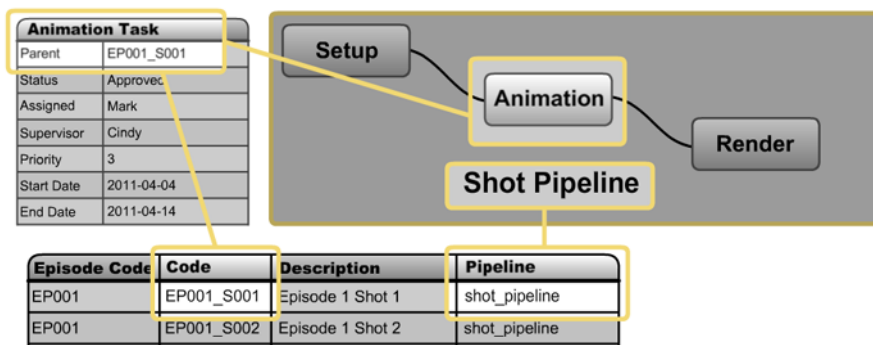
The layout of a Project Schema will depend on how the project is managed and how the sTypes relate to each other.



## Project Work-flow

The Project work-flow is a layout of a pipeline processes. Each pipeline defines a set of processes that a single object can travel through. These pipelines also represent the dependencies between the processes. For example, each process

inherently knows which processes are upstream, and which are downstream. This can be leveraged with automated notifications, emails, status updates and external trigger processing tools.



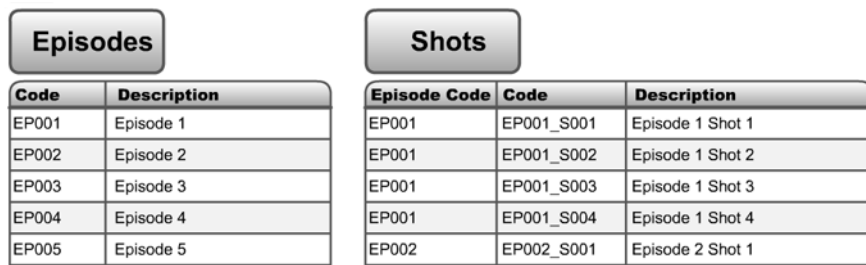
The Project Schema and the Project Work-flow Editor are connected. The Schema is used to layout the sTypes individually, and the Work-flow Editor is used to create the pipelines the sTypes will flow through.

## Searchable Types (sType)

What are "sTypes"?

Within the schema for a project there are various "types" of manageable objects that are defined when a node is created. These items are called Searchable Types (sTypes). Each of these Types are actually a table in the database and each column represents a property relevant to that sType.

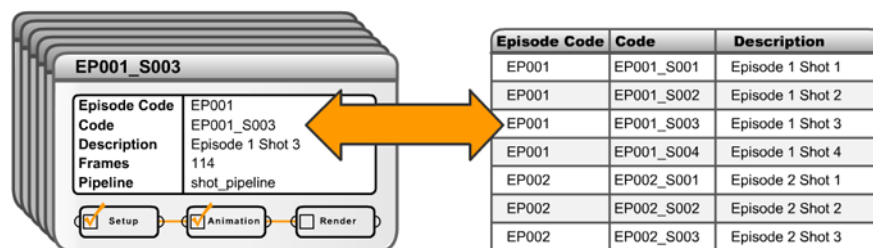
A project configuration can have various views, pipelines, naming conventions, access rules etc, which are all defined based on an assignment to a sType.



## Searchable Objects (sObjects)

What are **Searchable Objects (sObjects)** and how are they related to **sTypes**?

Searchable Object or sObjects are the entries in the sType's table. Each entry can be thought of as a 'container' or 'placeholder' for the object it represents ( a shot, an episode, a document etc). For example, Shot EP001\_S003 is the sObject, which we can see is an entry in the Shots table (sType) on the right.



## Common TACTIC sTypes

TACTIC provides a set of default sTypes:

- **Tasks**

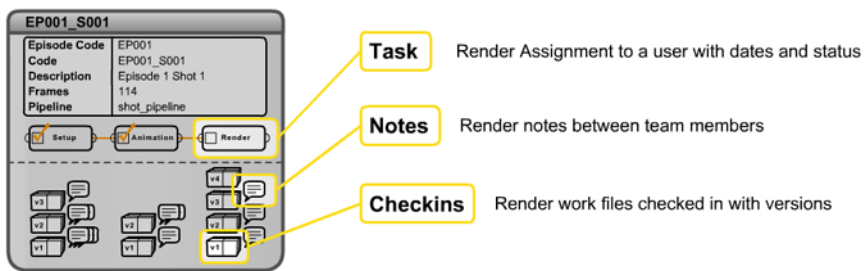
Tasks can be created based on the pipeline associated to a particular sObject. Tasks provide tracking for various processes such as; Status changes, Start and End date, Assigned User, Assigned Supervisor, etc. Users can be provided with a view that displays all assigned tasks, when each task is due, and what tasks to expect in the future etc.

- **Snapshots**

When you check in a file, it may involve one file or it might be 1000 files. A snapshot represents a complete package of that sObject at the point of check-in. Snapshots store the version and revision information, as well as the location of the file(s) in the file system. Snapshots can also store "dependencies" to other Snapshots. Dependencies provide a trail to indicate that one check-in is dependant on one or more other Snapshots.

- **Notes**

Notes can be added to any task or snapshot and provide instant feedback and real-time collaboration between end users. These Notes and update information can be sent to the user via TACTIC Notifications and/or email.



Within an object being tracked in TACTIC (shot, episode, document etc), separate child objects are stored and categorized by a 'process'. For example, the "design" process has notes, tasks, snapshots etc. All of these would be stored within the shot EP001\_S001. Through this concept, all of the life-cycle information regarding an object is easily accessible and organized and more importantly are used to drive the object through it's pipeline.

## TACTIC Interface

### Searching for Types of Objects

TACTIC interface allows users to search for information (sObjects) in the TACTIC system. End users are able to quickly find and display information relevant to their day-to-day tasks. The information can be displayed in a variety of ways such as simple tabular data, dynamic reporting or dashboards for example.

## Built-in STypes

When using the expression or adding widget config entries for built-in STypes, you would want to get familiar with how to represent notes (sthpw/note) or tasks (sthpw/task) for example. Below is the full list:

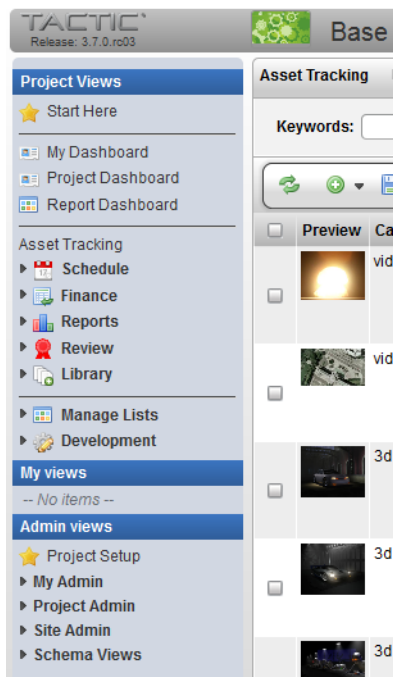
Name	SType
File	sthpw/file
Login	sthpw/login

## TACTIC Setup

<b>Name</b>	<b>SType</b>
Login Group	sthpw/login_group
Milestone	sthpw/milestone
Note	sthpw/note
Project	sthpw/project
Pipeline	sthpw/pipeline
Schema	sthpw/schema
Status Log	sthpw/status_log
Snapshot	sthpw/snapshot
Task	sthpw/task



## TACTIC Sidebar



### Description

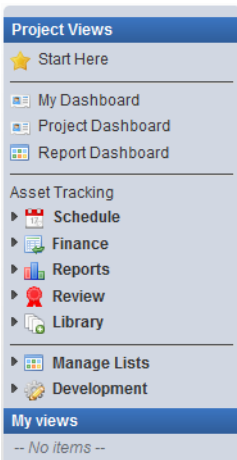
The TACTIC sidebar is the main menu system for navigating through the views of all TACTIC search types. The access rules applied to a specific account determine the contents of the sidebar as well as which views and search types are displayed when a user is logged in.

The items in the sidebar provide links to existing views of the different search types within a project. These views are built by your organization's production manager based on a selection of columns (properties), layouts (order and column width) and a search. If a search view is available, it provides a dynamic report based on the definition of the search.

Users at different levels can configure the sidebar to include only those views they need, or to include views that manage items and their relationships. For example, a user may want to set up a view where only the name, code and description of their own "storyboards" as in the view. Or, the user may set up a view where, for example, only those storyboards with a name containing the word "episode" and where child tasks have a status of "review" are in the view.

The sidebar is divided into three different categories, "Project Views", "My Views" and "Admin Views".

## Project Views and My Views



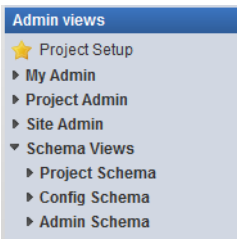
The Project Views provides a way to save project wide views that everyone across the entire project would want to see. It also has a manageable list of custom user views.

The Project Views can be defined by the person in the role of the project manager. Views can also be hidden from specific user groups.

**My View** contains a list of links to views that were created by the login user themselves. These usually are created by the user to cater to their own personal work flow.

## Admin Views

**Admin Views** displays the project schema and the TACTIC system and administration schemas. Access to the Admin Views section of the sidebar is generally reserved for admin level users.



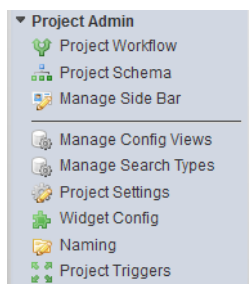
**Project Setup** - After initial creation of the project, this view contains the tools to setup the project: Create the Schema, Create Workflow, Manage the Side Bar.

**My Admin** - My Admin holds views that will allow the users to manage My Views and My Preferences.



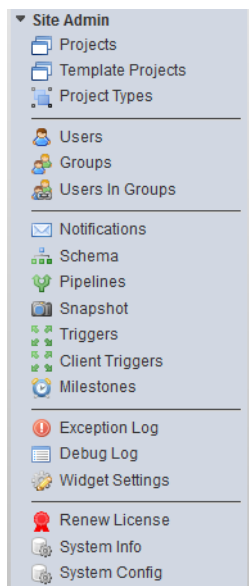
<b>Manage My Views</b>	Edits the views saved in the "My Views" section.
<b>My Preference</b>	Preferences include: Debug, Web Client Logging Level, Color Palette, Language, Quick Text for Note Sheet Thumbnail Size

**Project Admin** - Project specific views to manage the Project Workflow, Schema, Side Bar, Config Views, Search Types, Naming, Triggers.



<b>Project Workflow</b>	Workflow Editor for creating and editing processes and task status pipeline.
<b>Project Schema</b>	Schema Editor for creating and editing types and relationship connections.
<b>Manage Side Bar</b>	Edit the links and folders in the side bar.
<b>Manager Config Views</b>	Edit the asset view for each type.
<b>Manage Search Types</b>	Edit the columns for each type.
<b>Project Settings</b>	Set project settings such as use_icon_separation.
<b>Widget Config</b>	Look up and edit widget configuration by category, type, view name or key words in configuration.
<b>Naming</b>	Edit the automatic file naming and directory naming for checkin's.
<b>Project Triggers</b>	Edit the triggers by event, process, class name, script path, description, mode.

**Site Admin** - Site Administrator view to manage the Project, Templates, Types, Users, Groups, Users in Groups, Notifications, Schema, Pipelines, Snapshot, Triggers, Client Triggers, Milestones, Exception Log, Debug Log, Widget Settings, System Info, System Config



<b>Projects</b>	Edit the project info: preview, category, title, is_template, color scheme palette
<b>Template Projects</b>	Edit the project info for projects marked as template projects.
<b>Project Types</b>	Edit the project type info: dir naming cls, file naming cls, node naming cls, sObject naming cls, repo handler cls
<b>Users</b>	Edit the list of TACTIC users: preview, first name, last name, email, licence type

<b>Groups</b>	Edit the list of TACTIC groups: add group, users, description, global rules, access rules
<b>Users in Groups</b>	Drag and drop interface to assign users to one or more groups.
<b>Notifications</b>	Email notification configuration: email test, event, description, subject, message, group, rules, process, mail to,
<b>Schema</b>	Edit schema configuration, schema relationship connections.
<b>Pipelines</b>	Edit workflow pipelines: color, description, type, project code
<b>Snapshot</b>	Edit snapshots taken: preview, files, context, version, revision, login, description, is_current, is_latest
<b>Triggers</b>	Edit triggers: event, class name, script path, description, mode, project code
<b>Client Triggers</b>	Edit client triggers: event, callback, description
<b>Milestones</b>	Edit the list of milestone information: due date, lists tasks for that milestone, completion display
<b>Exception Log</b>	Lists all the exceptions when they occur: login, timestamp, class, message, stack traces.
<b>Debug Log</b>	Lists the debug log: category, level, message, timestamp, login
<b>Widget Settings</b>	Lists all the widgets and their settings.
<b>Renew License</b>	List TACTIC server license information and allow to browse for a new license: TACTIC version, who licensed to, max users, current users, expiry date.
<b>System Info</b>	Lists TACTIC server system information: server info, client, load balancing, mail server, asset folders, link test, python script test, clear side bar cache
<b>System Config</b>	Edit TACTIC Server configuration setup: Asset Management Setup, Mail Server, Look and Feel.

**Schema Views** - The schema view provides a hierarchical view of all of the search types in a project. The schema view can be a starting point when to create a project or user view.

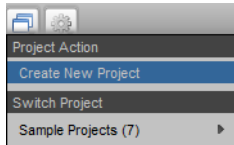


The **Admin Schema** appears in the schema sidebar and is accessible by users in the admin group. The Admin Schema provides access to types at the project and server level (e.g. users, groups, triggers and pipelines).

# Projects

## Create a New Project

To create a new project in TACTIC, load the Create Project wizard from the **Header Action Menu**.



Specify the project options in this wizard, then click **Create Project**.

 A screenshot of a 'Create Project' wizard dialog box. It has a title bar with 'Create Project', a close button, and a plus button. The dialog is divided into sections: 
 1. 'Project Title:' with a text input field and a help icon. Below it, text says 'The project title can be descriptive and contain spaces and special characters.'
 2. 'Project Code:' with a text input field. Below it, text says 'The project code is a very important key that will tie many components of the project together. \* Note: the project code must contain only alphanumeric characters [A-Z][0-9] and only an '\_' as a separator'.
 3. 'Project Image:' with a 'Browse' button. Below it, text says 'The project image is a small image that will be used in various places as a visual representation of this project.'
 4. 'Copy From Template:' with a dropdown menu showing '-- Select --'. Below it, text says 'This will use the selected project template as a basis and copy all of the configuration elements. Only template projects should be copied.'
 At the bottom, there is a checked checkbox labeled 'Jump to New Project' and a 'Create >>' button.

### Create Project Options

<b>Project Title</b>	The project title is displayed many places in the TACTIC interface, so it should be the full name of the project for all users.
<b>Project Code</b>	The Project Code is automatically generated once the Project Title has been entered. This code should only contain alphanumeric characters or the underscore "_" character. This code is used to tie all elements of the production together. For this reason, a project code name cannot be changed after it is set up and saved through this wizard.
<b>Copy From Template</b>	This option facilitates and speeds up the creation of a new project from an existing project. It creates a new copy of all your project records to the new template project, such as asset libraries, templates, and configurations. Once the new project has been created, the admin user can then make any adjustments to their new project. Please note that only template projects should be copied.

## Accessing Projects from a Web Browser

The new project can be access from the web browser using the URL:

**http://(server)/projects/(code)/**

For example, for a project with the code "test" on a server called "tactic", the URL would be:

**http://tactic/projects/test/**

Once the new project is created, navigate to the **Admin views > Project Setup** link in the sidebar.

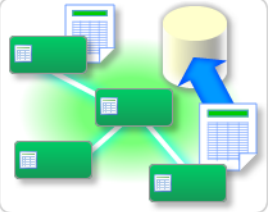
The Project Setup view provides some guidance on how to set up a project.

Project Setup

Project Setup

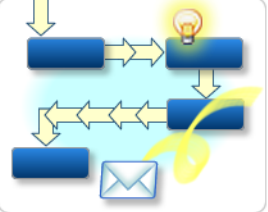
To set up a TACTIC project, follow the below steps. Begin by creating the schema, then create pipeline(s) for each sType (searchable type) that requires one. The Side Bar can then be set up to provide specific views to various types of users of the project.

Create Schema



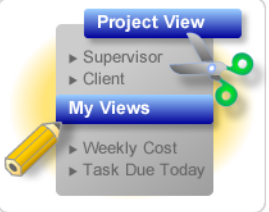
The schema is a collection of nodes that layout the basic components of a project. Each node represents a separate list of items (sType) used in this project: ie Assets, Shots, Artwork, etc.

Create Workflow



Pipelines define how particular items of an sType will move through its lifecycle. Creating pipelines will also allow you to set up automatic triggers and notifications for each process.

Manage Side Bar



The Side Bar can be easily configured to show specific views of your project to each user.

## Can a New Project Be Undone?

Unlike most operations in TACTIC, when you create a project it is not easily undo-able because of the number of complex actions that must take place:

1. It creates the database for the project and copies the schema designated for this type of project.
2. It creates project sites in the sites directory.
3. It registers the site in TACTIC.

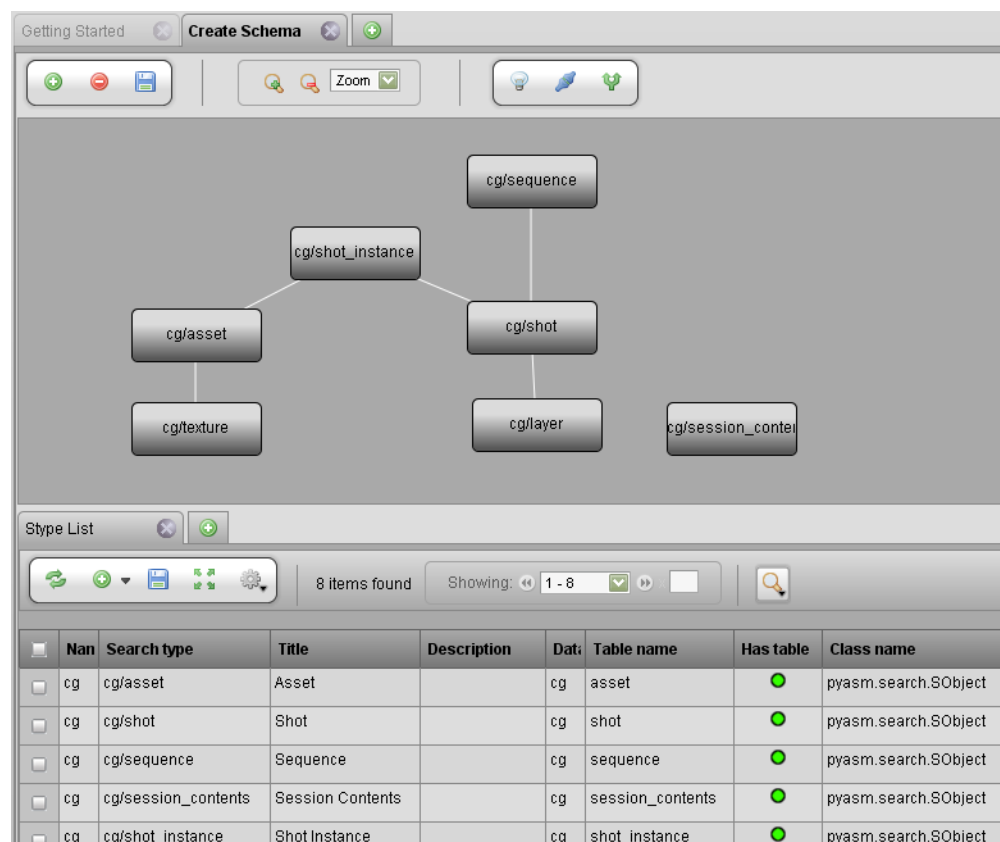
If a project is accidentally created, then it is best to delete the project immediately.

15

# Projects

## Project Schema

The project schema is used to create structure or a "data model" of a project. The Schema view defines the type of items managed by using a visual graphical node editor. The Schema Editor displays the layout of the created sTypes and the connections between them.

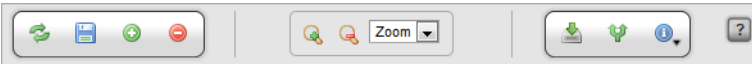


The Project Schema Editor is available through the Getting Started link in the side bar which is available after creating a project, or under the Admin Views under Project Admin -> Project Schema in the side bar.

The Project Schema editor is an essential tool used for the creation of new project templates. This editor is used to layout the various types of objects (files, assets) that will be managed and produced on a project. These types (sTypes) are searchable within TACTIC. Node based layout and work-flow, allows for simple manipulation and creation of these various sTypes and their relationships to each other.



## Editor Button Shelf



### Main Editor Buttons

<b>Add</b>	Add a new node to the canvas. This represents an unregistered sType
<b>Delete</b>	Delete the selected nodes or connections from the canvas
<b>Save</b>	Save all changes to the schema

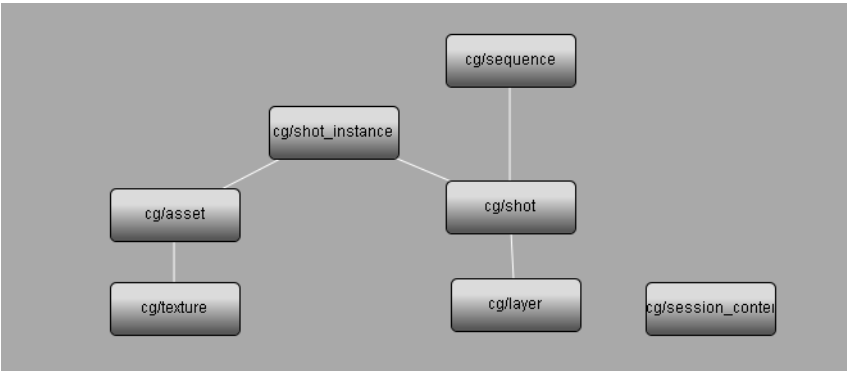
### Editor Zoom Controls

<b>Zoom In</b>	Zoom the canvas in
<b>Zoom Out</b>	Zoom the canvas out
<b>Zoom Options</b>	Allow for choosing the zoom level.

### Node Options (Applies to the selected nodes or connections)

<b>Register sType</b>	Registers the selected nodes as new Searchable Types using the registration wizard. If more than one node is selected, the sTypes will be registered in batch.
<b>Edit Connection</b>	Load the connection editor pop-up.
<b>Edit Pipelines</b>	Load the Project Work-flow (pipeline) editor.

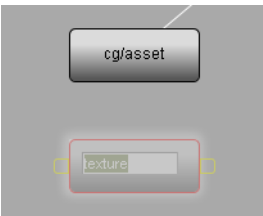
## Laying out the sTypes



To create a new Searchable Type (sType) in the schema, add a new node to the canvas using the [+] button in the editor. It's also possible to create a new node from an existing node by simply dragging a connection line from the output handle of the existing node.



Once the type has been created on the canvas, it can be renamed by right clicking on the node or using a "CTRL-click" on the node.



### Note

It is important to note that during this initial process, you are creating a "blueprint" for your project. The next steps are to **register** the sTypes. Each sType in TACTIC is represented as a table in the project database, this table is required to go through a registration process. This process will generate the table as well as provide the opportunity to add columns (properties), a pipeline, default views for the sidebar and more.

## Workflow (Pipelines)

Where applicable, you can add the pipeline attribute to a search type to allow for association of the sObjects to a particular pipeline. Having a pipeline assigned allows an sObject to travel through a set number of processes. For each of these processes, a task can be created and assigned to a user, files can be checked in, notes can be submitted and work hours can be logged.

By choosing "Has Pipeline" on creation, an extra "pipeline\_code" property will be added to store pipeline associations and a Pipeline will be created and registered for the new sType.



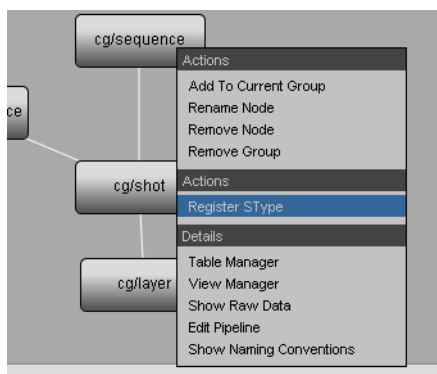
### Note

To edit the pipeline, you can click the pipeline link in the top of the editor or, in the sidebar navigate to Project Admin -> Project Workflow.



## Node Options

Once registered, each node provides options for further configuration of sType related project setup and configuration, which can be executed through the main shelf buttons or by right-clicking on a node:



### Editor Actions

<b>Add to Current Group</b>	Adds the selected node(s) to the current group (pipeline)
<b>Rename Node</b>	Rename the node (sType)
<b>Remove Node</b>	Remove the node (sType)
<b>Remove Group</b>	Removes the group (pipeline)

### Node Actions

<b>Register sType</b>	Loads the sType registration wizard
-----------------------	-------------------------------------

### Node Options

<b>Table Manager</b>	Load the Database table manager for the selected type ( <i>see "Table Manager" below</i> )
<b>View Manager</b>	Loads the view manager for the selected sType ( <i>see "View Manager" below</i> )
<b>Show Raw Data</b>	Loads the Raw database data in a table for the selected sType ( <i>see "Raw Data" below</i> )
<b>Edit Pipeline</b>	Loads the Workflow Editor allowing access to edit the pipelines related to the selected sType.
<b>Show File Naming</b>	Loads the file naming table for the selected sType ( <i>see "File Naming" below</i> )

## Table Manager

Stype List **Table [cg/asset]**

**SType Table Manager**

SType: **cg/asset**

-- Action --

Database Columns

**Db column**

- Id
- Code
- Pipeline Code
- Login
- Timestamp
- S Status**
- Name
- Description
- Asset Category Code
- Asset Type

**Column Detail**

Mode: **advanced**

Column Definition

Name: **asset\_type**

Data Type: **varchar**

☒ Allow null(empty) value:

**Modify Column** **Delete Column**

Advanced - XML Column Definition

## View Manager

Stype List **View [cg/asset]**

Search Type: **cg/asset (Asset)** View: **table**

**Table**

- Preview
- Code
- Name
- Description
- Asset Category Code

**Name: name** Mode: **Form**

Element Attributes

Title:

Width:

Editable: ☒

## Raw Data

Stype List **Raw Data [cg/asset]**

4 items found Showing: 1 - 4

	Code	Name	Description	Asset category code	Asset type	Pipeline
<input type="checkbox"/>	bridge	Bridge		Envrinment		
<input type="checkbox"/>	chr001	name	description text			
<input type="checkbox"/>	creature	Creature		character		

## File Naming

Stype List **Naming [cg/asset]**

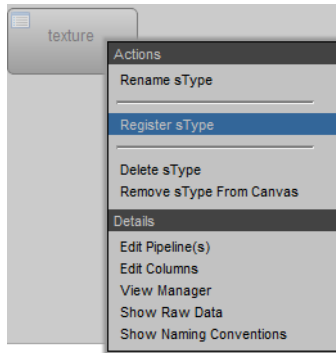
1 item found Showing: 1 - 1

	Search t	Cor	Sn	Dir naming	File naming	Sar	Lat	Cur	Mar
<input type="checkbox"/>	cg/asset			assets/{subject.asset_library_code}/{subject.code}/{context}	{subject.code}_{context}_{version}.{ext}				<input checked="" type="checkbox"/>

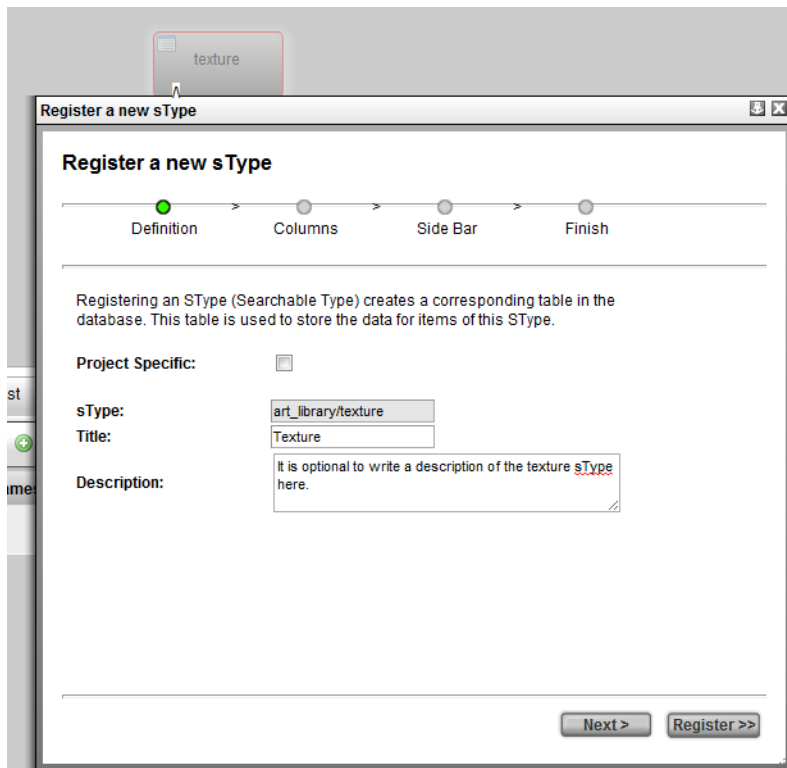
## Register sTypes

sTypes can be registered one at a time, or in a batch format. The benefit of registering each sType individually is the opportunity to configure and select properties of the new sType that are outside of the TACTIC defaults.

To register an sType, right click on the node to call the menu and choose "**Register sType**"



The **Register a new sType** wizard will appear:



### Definition Information

<b>sType</b>	Refers to the database name for the sType. in a "<project>/<name>" format. If no project is defined (i.e.. "art/") then the current project namespace will be used.
<b>Title</b>	The title for the sType is used in the UI for display purposes.

<b>Description</b>	An optional description of the sType.
<b>Project Specific</b>	

Once the Definition information has been entered, press "Next" or press "Register" to complete the registration process. Note: It is recommended to go through the series of steps outlined in the "Register a new sType" wizard, as this allows for quick and easy configuration of the new sType that is outside of the TACTIC defaults.

## Columns

**Register a new sType**

Definition → **Columns** → Side Bar → Finish

All sTypes can have pipelines which allow a pipeline defined in the project workflow to be assigned to an object.

**Has Pipeline?** ☒

All sTypes can have icons associated with them.

**Include Preview Image?** ☒

Extra attributes can be added here. These are direct columns to the corresponding table of the sType. Each column will be mapped directly to an attribute of the sType.

A number of columns are created by default for every sType. These include: id, code, name, description, login and timestamp.

**Add Columns to sType:**

Name:  Type:  (+) (-)

< Back      Next >      Register >>

<b>Has Pipeline</b>	If selected, it will automatically set up an association for a pipeline workflow for the sObjects in that sType. <i>The section below describes this relationship in more detail</i>
<b>Include Image Preview</b>	The Preview image column will be added to the sType List view
<b>Add Columns to sType</b>	During the registration process, default columns are added to the new sType table. You can also add additional columns during this process. <i>Note - columns can be added after this process using the <b>Table Manager</b></i>

## Side Bar

The Side Bar registration page adds default views that can be useful to a new project. These views include forms for scheduling tasks, managing lists and adding new items (sObjects).



texture

**Register a new sType**

Definition Columns **Side Bar** Finish

These predefined views for the newly registered search type will be added to the side bar. They can be modified under "Manage Side Bar"

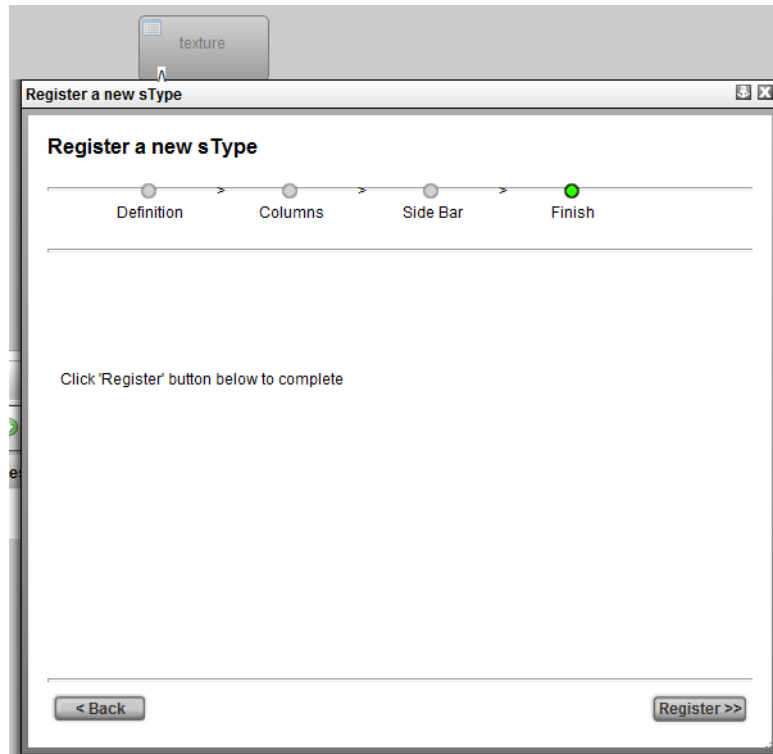
**Add Predefined Links in Side Bar:**

- ☒ Content List
- ☐ Add Content Form
- ☐ Task Schedule
- ☐ Tracking

< Back Next > Register >>

<b>Content List</b>	
<b>Add Content Form</b>	
<b>Task Schedule</b>	
<b>Tracking</b>	

**Finish**

**Finish**

To complete the registration process, press "Register". It is also possible to navigate back to previous stages by pressing the "Back" button.

## Connecting sTypes

In a project, items (ie. files, assets) may be related to each other. For example, a car is built with various parts that can be identified separately but are all related to the same car. Another example can be found cinematic film production. The cinematic footage of one movie is commonly broken down into sequences and shots.

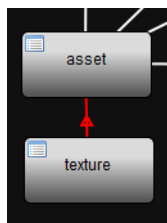
**How do these relationships work in TACTIC?** - Each sType is represented as a table in the database and each entry in the table represents an sObject. The relationships are created when storing matching data "properties" in each of the tables. In the example tables below there are "Sequence" and "Shot" sTypes. The "code" column matches the "sequence\_code" column which illustrates which shot is related to which sequence.

code	description
SEQ001	The first sequence
SEQ002	The first sequence

sequence_code	code	description
SEQ001	SEQ001_001	Sequence one shot one
SEQ001	SEQ001_002	Sequence one shot two
SEQ002	SEQ002_001	Sequence two shot one

In the Schema Editor, relationships are represented by lines connecting the nodes. When these connections are made, the columns used to relate the sTypes can be chosen in the Connection Editor.

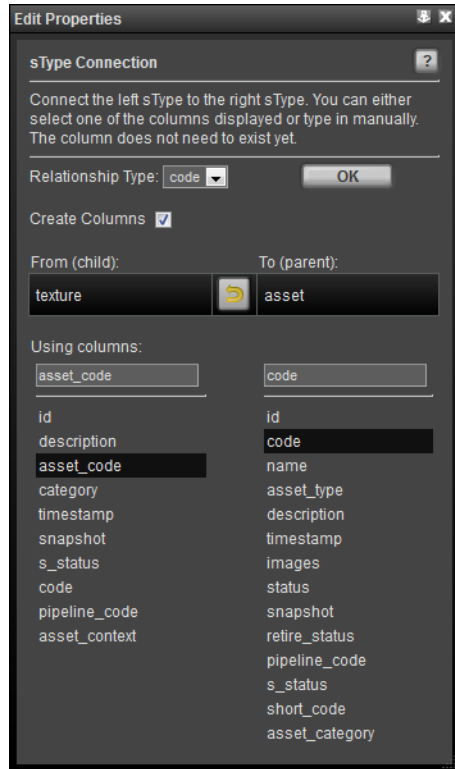
To create a new connection, hover over a node and click-drag a connection to the desired node (sType).




### Note

The direction of the arrow in the connection indicates from child to parent.

After a connection is made, the Connection Attributes editor will open to enable the choice of column relationships. It is also possible to create new columns from this editor.



## Note

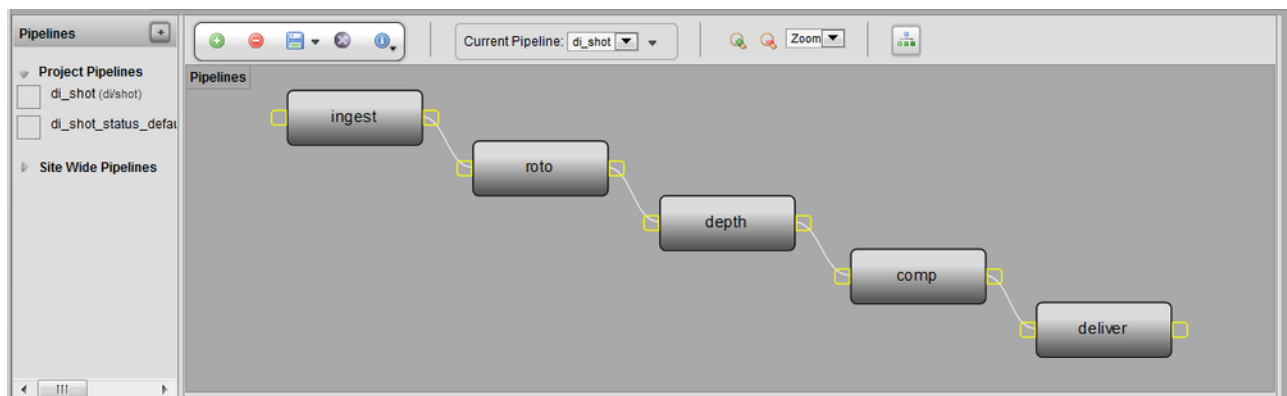
The yellow **Switch** button  in the middle of the tool toggles which node is the child and which one is the parent.

# Projects

## Project Workflow Introduction

The **pipeline** defines all of the processes and deliverables required in the creation of an item, and is the central entity that ties the whole production together. In simple terms, a pipeline is simply a definition of workflow processes.

In TACTIC, the pipeline workflow is used to lay out the steps which a particular Searchable Type (sType) needs to follow as it flows through its processes. Searchable Objects (sObjects) in TACTIC are by nature like static containers or place holders that contain everything that relates to them. When this content needs to be organized, managed and produced as a part of a workflow, this is where a pipeline comes into play. A Pipeline allows for this item "container" to be placed on a digital conveyor belt where it will stop at each process and will be filled with Tasks, Notes, Snapshots (checked in files) and more. On top of this, the contents are tagged with this process allowing for a separate history representing each stop on the conveyor belt.



## Workflow Concepts

At the very beginning, a clear diagram should be defined of the processes, their relationships to each other and the deliverables between each of the processes.

Each **sType** can have its own pipeline(s), so you create different pipelines for different sTypes.

The best approach to building a pipeline is to start simple, processes can always be added later. Overall, the general concept for defining the processes in a pipeline, is to break down each place where separate file versioning, tasks and notes will need to be generated and tracked

Simply put: you have a series of processes, named in any way you wish. Each process is often represented by a task assigned to a user which needs to be worked on. The completion of a process occurs when this task complete (often indicated by setting it to a final status ie approved, complete etc). While this task is in progress, files will be checked in and notes will be tracked as the process is worked on.

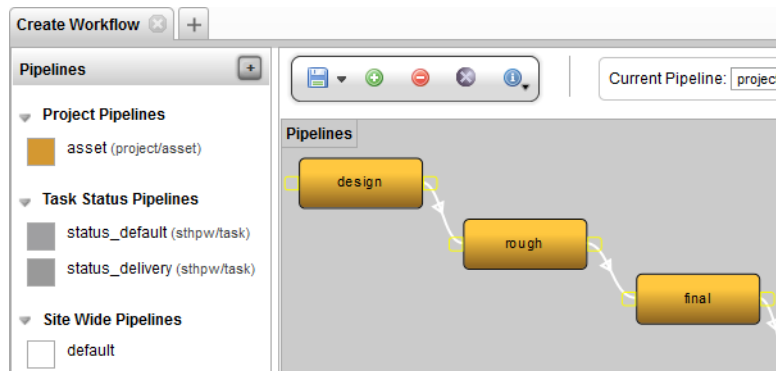
## Status

Each process also contains a set of possible "statuses" which which typically a used by the tasks. For example the "model" process can have a possible status of (Waiting, Ready, In Progress, Revise, Review, Approved, Client approved). Each status helps track the current state of the process and are often the spawning point of automatically setting downstream and upstream process status, sending automated notifications and using the python triggers, etc.

## Subcontext (advanced)

At times there may need to be a further breakdown within a process, this can be achieved through using a **subcontext**. Sub-contexts are used for departments to check-in and track work and progress internally without other departments needing to interact with the content. For example, in the case of a VFX process in a shot pipeline, someone checking in files may be able to add extra specification (subcontext) such as VFX/dynamics, VFX/water, or VFX/smoke. Another situation is where multiple variations of something need to be checked in. For example you may need a "red" and a "blue" design so the subcontexts for a design checkin would be design/red and design/blue.

# Workflow Editor



## Introduction

The Workflow Editor is a graphical tool in TACTIC used to interactively create pipelines (workflows). It is a node-based tool which creates processes in a pipeline and connects them. The Workflow Editor makes it easier to create large complex pipelines to filter and process information and file system flow.

The Workflow Editor is simple to use and similar to node base utilities commonly found in other applications. Nodes can be created in the canvas and connected together. Each node represents a process (with attributes associated to it) and each connection represents information being delivered from one process to the other. Together, the Workflow Editor helps you create a definition of the pipeline document and drive much of the information flow in TACTIC.

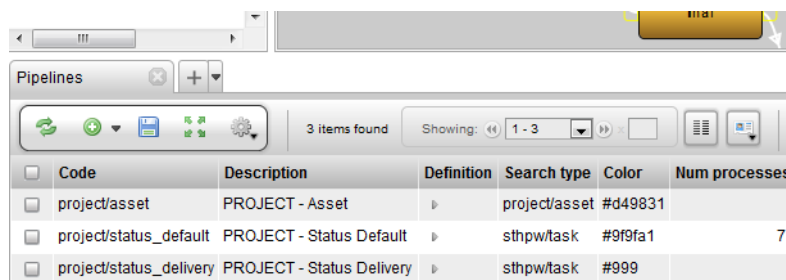
### Access the Workflow Editor

Access the Workflow Editor by going to:

**Admin Views -> Project Admin -> Project Workflow**

## Insert a new pipeline

When the option for "Has Pipeline" is selected during the registration of the sType, this defines a default pipeline for that sType. This pipeline can be found defined in the Workflow Editor in the sidebar under **Project Pipelines**. To add a new pipeline manually, the select the [+] icon in bottom panel of the Workflow Editor.



## Interface Walk Through

The buttons at the top of the Workflow Editor allow various operations on the canvas:



- **Create:** Creates a new node on the canvas.
- **Delete:** Deletes the selected node.
- **Save:** Saves the current state of the pipeline to the database.
- **Clear:** Clears the canvas.
- **Properties:** Opens the Node Properties panel.

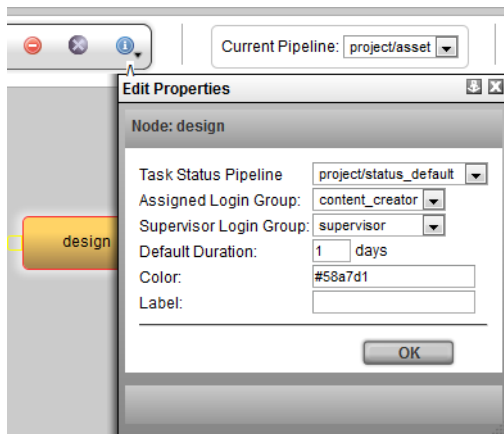
## Edit Properties of a Pipeline

To edit the properties of a pipeline, first select a node in the pipeline and then click on the **Edit Properties** button on the tool shelf.



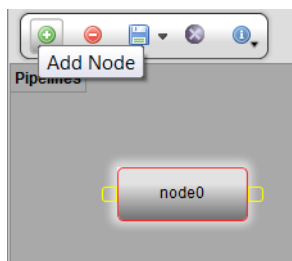
### Note

For more information regarding the Process Options, refer to the section Project Workflow -> Pipeline Process Options



## Lay Out a Pipeline

When you click the green plus button, **Create** , a new node will appear on the canvas.

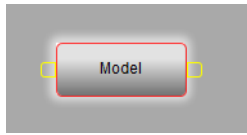


Rename the node: Select the new node and press CTRL-LMB to rename the node. Alternatively, right click and select **Rename Node** from the context menu.

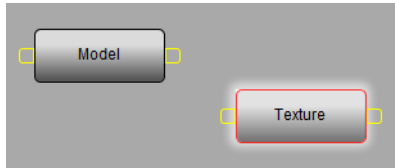




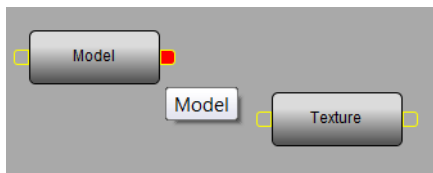
Type in the new name for the node ("Model," in this example), and press **Enter**.



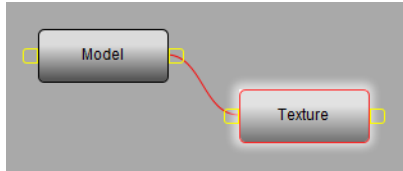
Create another new node (called "Texture" in this example).



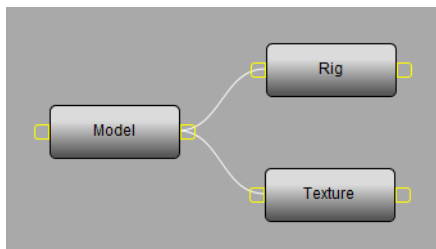
To create a connection between the two nodes, click on the handle on the right side of the "Model" node. This will create a connector which will follow the cursor.



Click on the left handle of the "Texture" node to complete the connection. Now, the 2 nodes are connected together. Once 2 nodes are connected, they will stay connected unless the connector is selected and deleted.



It is also possible to have one node connect to more than one node. In the following example, the "Model" process delivers to both the "Texture" process and a "Rig" process:



## Pipeline Workflow Automation

Repetition and daily components that make up a user's workflow can be made easier through automation of notifications, file/directory naming and triggering custom logic. Automations such as these can vary from simply sending an email or automatically setting upstream and downstream task statuses to running custom Python scripts and tools to encode files, submit renders, generate previews, deliver files to clients, etc.

On the Workflow Editor's canvas, right-clicking on a node will bring up the context menu where the automation interfaces can be loaded into the lower half of the interface. These options include:

<b>Show Properties</b>	Loads the Node Properties window.
<b>Show Triggers/Notifications</b>	Loads the Triggers and Notifications setup Interface
<b>Show File Naming</b>	Loads the Directory and File naming convention setup Interface



## Note

Each of the menu options are explained in the "Project Automation" section of the documentation.

## Mouse and Keyboard Shortcuts

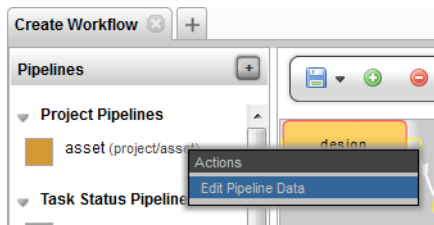
When the cursor is over the canvas in the pipeline editor, the following mouse and keyboard shortcuts are available:

<b>LMB</b> on a node	Select the node
<b>LMB</b> on the empty canvas space	Clears selection
<b>LMB + Ctrl</b> click on a node	Edits the name of the node
<b>LMB + Shift</b> click on a node	Add node to selection
<b>LMB + drag</b> on a node	Drags the node around the canvas
<b>LMB + drag</b> on the empty canvas space	Pans around the canvas
<b>LMB + Shift + drag</b> to form a selection box	Forms a selection box
<b>LMB + Ctrl + drag</b> to the left or the right	Zooms in or out on the canvas
<b>DELETE</b>	Deletes the selected node(s)

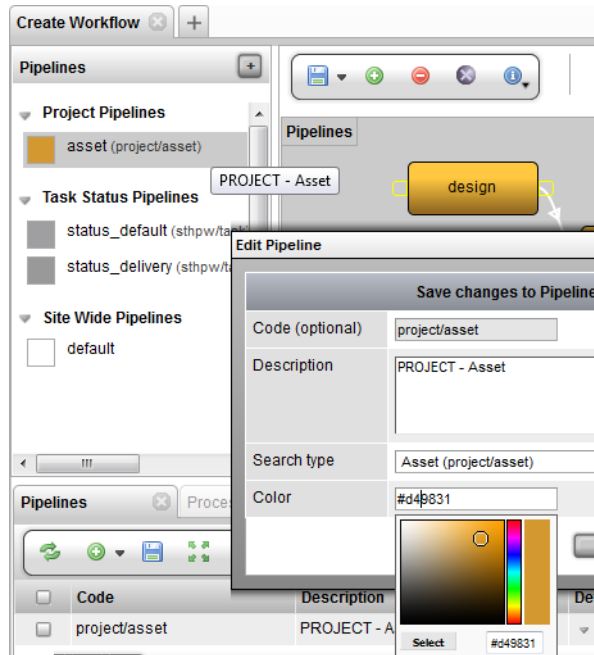
## To Change Node Color

To change the node color, go to the **Workflow Editor -> sidebar**

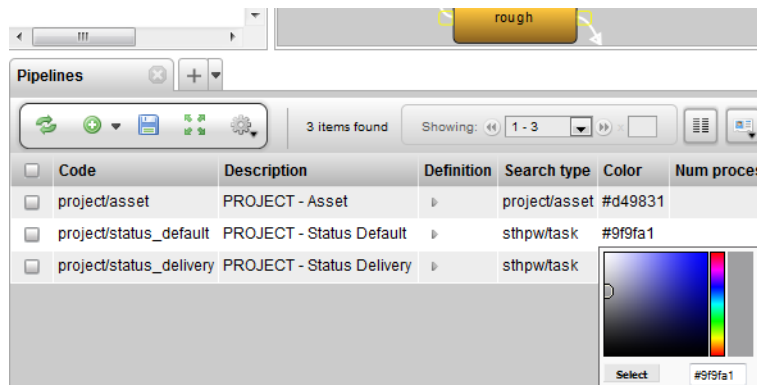
right click on the pipeline and select **Edit Pipeline Data**



Next, click on the color input field. A color swatch will pop-up. Select the new color for this pipeline from the color swatch.

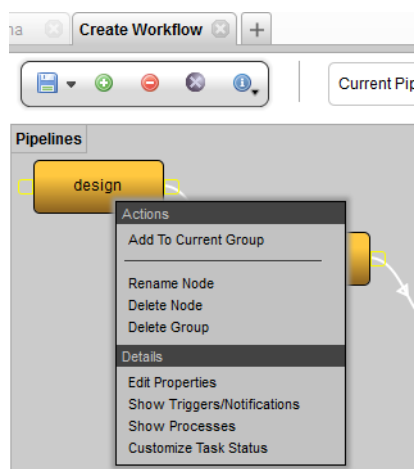


Another way to change the color is in the **Workflow Editor -> Pipelines** tab (panel at the bottom) click on the **color** column and pick the color from the color swatch.



## Pipeline Node Context Menu Options

Right click on the pipeline node will display the following menu options:



<b>Add To Current Group</b>	Add the selected node to the current group
<b>Rename Node</b>	Rename the current selected node
<b>Delete Node</b>	Delete the current selected node
<b>Delete Group</b>	Delete the group for the current selected node
<b>Edit Properties</b>	Edit the properties for the current selected node
<b>Show Triggers/Notifications</b>	Display the triggers and notifications view in the bottom panel
<b>Show Processes</b>	Display the processes in the bottom panel
<b>Customize Task Status</b>	Create a custom task status pipeline for the process (refresh the Workflow Editor to see it added to the sidebar)

## Advanced

Behind the scenes, the pipeline is an XML text document. This document is how TACTIC stores its representation of the pipeline structure of nodes and connections.

Although it is rare to need to manually edit the pipeline XML structure, it is available at the bottom of the Workflow Editor in the pipelines table in the **Data** column.

Below is an example of the pipeline XML for the **Model -> Rig / Texture** pipeline:

```
<?xml version='1.0' encoding='UTF-8'?>
<pipeline scale='100'>
  <process name='model' ypos='-95' xpos='-138' />
  <process name='rig' color='blue' xpos='38' completion='80' task_pipeline='task' ypos='-165' />
  <process name='texture' ypos='-51' xpos='42' />
  <connect to='rig' from='model' />
  <connect to='texture' from='model' />
</pipeline>
```

## Pipeline Process Options

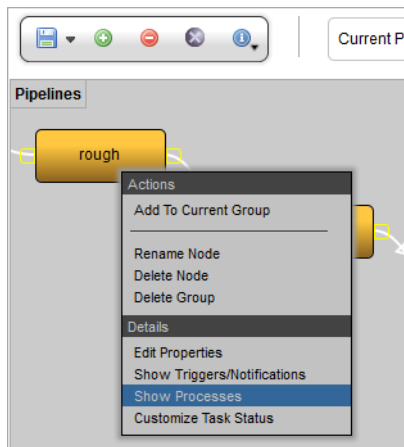
Each node has a number of properties that can be set. These properties may be used by TACTIC to derive useful information. These properties are:

<b>Task Pipeline</b>	<b>Status</b>	Selects from a list of 'Task Status Pipelines' to connect the selected process to. This accommodates a separate set of statuses for the specific process. These pipelines are defined the same way other pipelines are. The only difference is that these pipelines are assigned to the sthpw/task sType. This property represents the "code" property of the task pipeline.
<b>Assign Group</b>	<b>Login</b>	Specifies the process to a particular group of artists.
<b>Supervisor Login Group</b>		Specifies the process to a particular group of supervisors.
<b>Default Duration</b>		Set the a duration schedule (in days) of the process.
<b>Color</b>		The color to represent the process in the GUI. For example the Task Status Widget can be setup to display the color of the process.
<b>Label</b>		Add a label for the process.

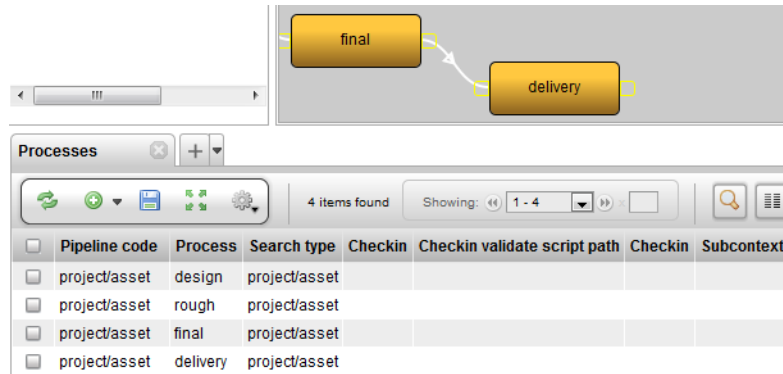
To open the **Edit Properties** pop-up, select a node and then click on the **Properties** button on the tools shelf:

Further process options can be found by right clicking on the node in the:

### Pipeline Editor -> Show Processes



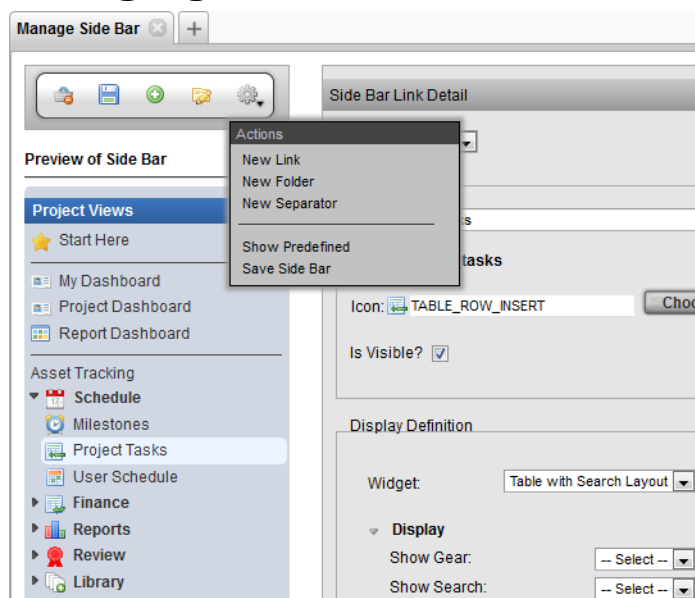
The **Processes** tab will appear in the panel at the bottom:



<b>Pipeline Code</b>	eg. project/asset
<b>Process</b>	eg. design, rough, finale, delivery, etc.
<b>Search Type</b>	eg. project/asset
<b>Checkin Mode</b>	File   Directory   Sequence   Multiple Files
<b>Checkin Validate Script Path</b>	Path to a script which is run upon checkin for validation.
<b>Checkin Options View</b>	Advanced custom layout to be used for checkin view.
<b>Subcontext</b>	eg. hi_res, low_res, etc.

# Interface Setup

## Managing the Sidebar



### Introduction

The sidebar is a menu of views created by the administrator to present information on the items in TACTIC. Examples of views include: Asset Tracking, User Schedules, Milestones, Project Tasks, Expenses List, Budget, File Usage Report, Burndown Report, etc.

The tool to manage the sidebar can be found under:

**Admin Views -> Project Admin -> Manage Sidebar**

The Manage Side Bar view is divided into 3 panels:

- **Tool Shelf** - Quick links to access sidebar tools.
- **Preview of Sidebar** - Re-arrange the elements in the sidebar by dragging and dropping.
- **Element detail** - First, select a link from the preview of the Side Bar. Then, this panel allows for editing of the link element's properties.

### Preview of Side Bar

The Preview Side Bar panel allows for changes to be made and tested before committing to the actual sidebar. The following is a list of actions that can be carried out in the preview panel:

- Drag Links into folders
- Rearrange Links and Folders
- Rearrange separators
- Drag links and folders into the trash for removal

- Selection of a link or folder to edit the properties or security

## Side Bar Link Detail

When editing the properties of a link, at the bottom of the Side Bar Link Detail panel are security settings. This section provides the opportunity to select which groups can see and the folder or the link. Changes to security to other users take effect when those users refresh their sidebar.

Group	Group default
<input checked="" type="checkbox"/> admin	allow all
<input type="checkbox"/> client	allow all
<input type="checkbox"/> content_creator	allow all
<input type="checkbox"/> executive	allow all
<input checked="" type="checkbox"/> supervisor	allow all

Save

The security section provides a method of simplifying what is presented to team members of different user groups. The security settings behaves in a hierarchical manner. If a specific security is applied to a folder, its child links will inherit the same security setting. The security settings are applied at runtime (i.e. the child link does not have the security saved to it).

The Side Bar Link Detail panel allows for editing of Link properties in either simple or advanced mode.

After choosing the groups which should access a link, click the Save Definition to apply the security.



## Note

The security configuration settings are saved as XML access rules and can be found under: **Admin Views** -> **Site Admin** -> **Groups**

## Simple Mode

Side Bar Link Detail

Mode: simple

Attributes

Title: Project Tasks

Name: project\_tasks

Icon: TABLE\_ROW\_INSERT Choose

Display Definition

Widget: Table with Search Layout

Display

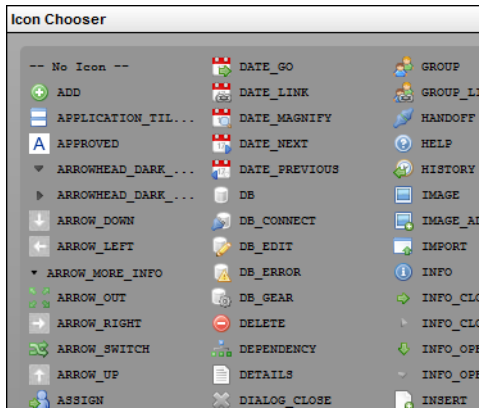
Show Gear: -- Select --

Show Search: -- Select --

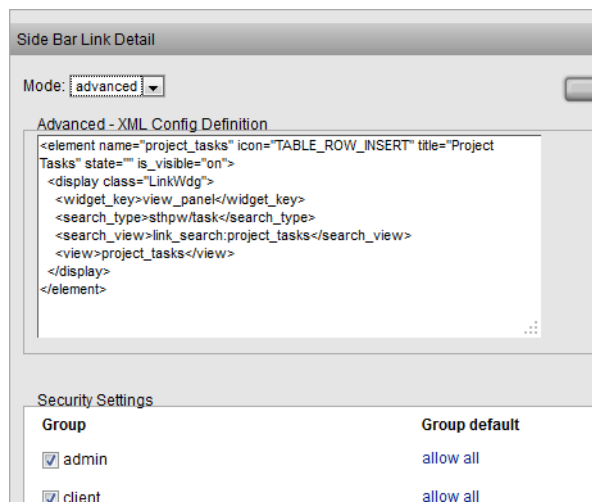
In simple mode, the following aspects can be edited:

- **Title:** The title of the Link.
- **Icon:** An icon can be selected in the interface below by clicking the **Choose Icon** button





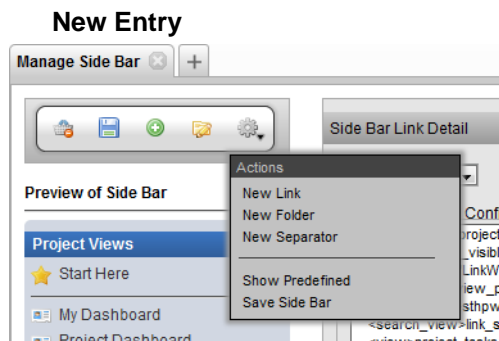
## Advanced Mode



Advanced mode shows the raw XML used to configure the Link Element. The element is saved to the definition view for the sidebar.

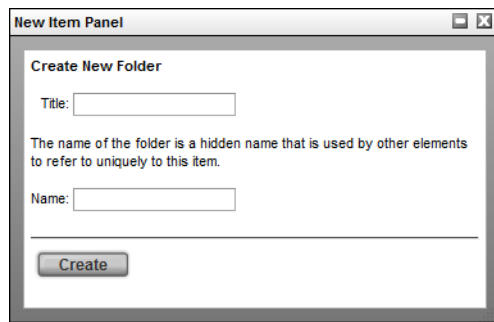
## Action Menu

The project views manager **Action Menu** provides links to tools for managing the sidebar.



The Add new link option adds a new link to the sidebar which is linked to the default table view for the chosen Search Type

## New Folder

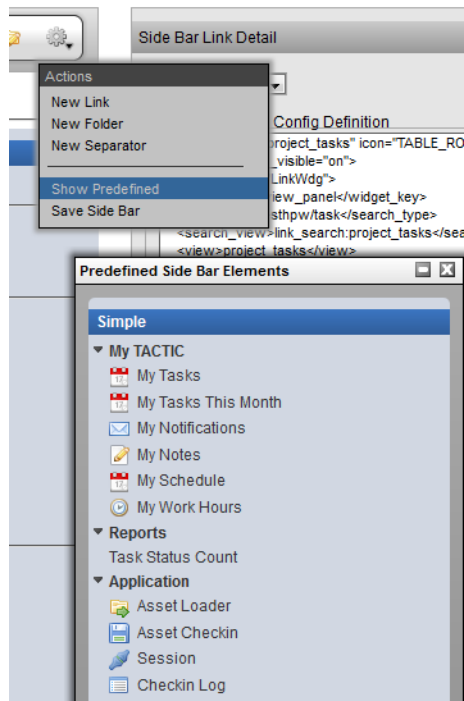


Adds a new folder to the sidebar. Items can be dragged into the new folder in the pop-up then saved or, Items can be dragged into the folder at anytime in the editor.

## New Separator

Adds a new separator to the sidebar, these can be used to further organize the folders and links

## Add Predefined

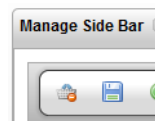


The predefined views are delivered as part of a project module. These predefined project can be utilized to expedite the setup process.

If the project is a custom (simple) project, only the **My Tactic** views will be available.

## Save

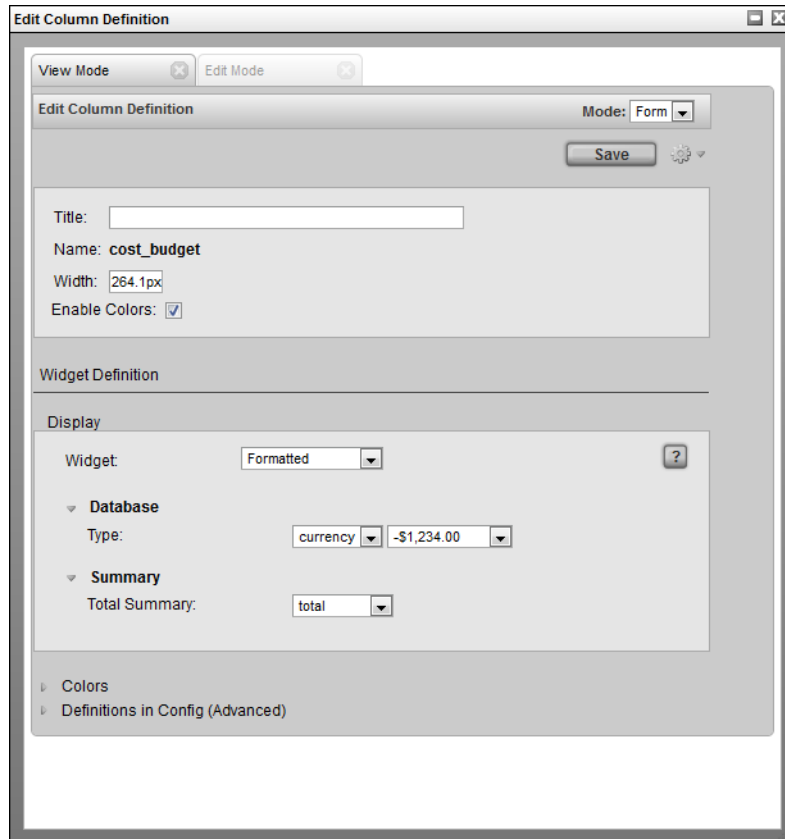
Saves the current state of the Temp side bar. Clicking the save icon



will also save the state.

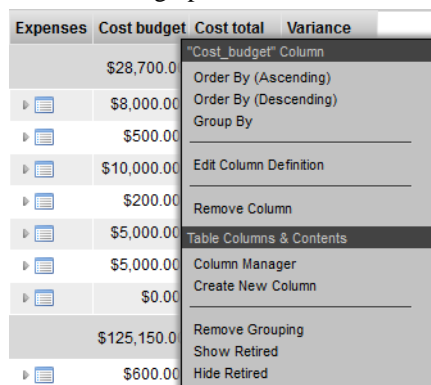
## Element Definition Widget

Editing the configuration of widgets is an important part of configuring TACTIC. Widgets are drawing elements that display information on the TACTIC interface. Widgets can be configured for a wide variety of applications. The Element Definition Widget allows for the generic configuration of any widget using an easy to use interface directly in TACTIC.



## Accessing the Element Definition Widget

The Element Definition Widget can be used to edit existing widgets or to create entirely new ones. It can be accessed from a few locations. The common way to access the widget is by right clicking on the column header in the table. This will bring up the context menu:



Selecting **Edit Column Definition** opens a pop-up with the appropriate data filled in for the selected element.

**Edit Column Definition**

View Mode Edit Mode

Mode: Form

Save

Title: Checkin

Name: general\_checkin

Width: 27px

Enable Colors: ☒

**Widget Definition**

Display

Widget: -- Class Path --

> Class Name: tactic.ui.table.CheckinButtonElement/Wdg

**1. Required**

Transfer Mode: -- Select --

**2. Display**

Mode: -- Select --

Checkin Script Path:

Validate Script Path:

Process:

Lock Process: ☐

Width: 600

Show Context: -- Select --

Show Sub Context: -- Select --

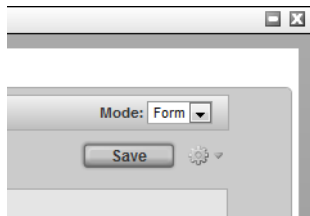
Colors

Definitions in Config (Advanced)

The "Create New Column" selection opens up an empty Element Definition pop-up so that new elements can be created.

## Tool Bar

The tool bar can be found in the top right hand corner of this widget.



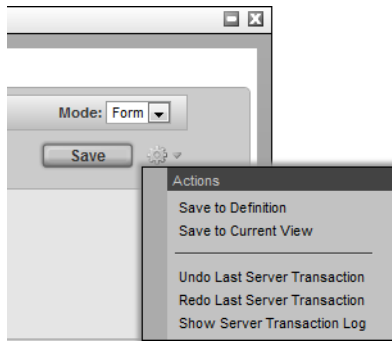
**Mode:** This can be set to either **Form** or **XML**. The **Form** selection is the default which displays the user interface for entering in attributes for this widget. The **XML** section is for advanced usage which allows direct control of the XML definition of the widget.

**Save:** This button will save the settings in the widget to the definition view.

**Gear:** Clicking on the gear menu will display a number of other options available as described in the next section.

## Gear Menu

The gear menu contains a number of operations.



**Save to Definition:** This will save the current contents to the "definition" view. The definition view is a view where all widgets for a particular sType are defined.

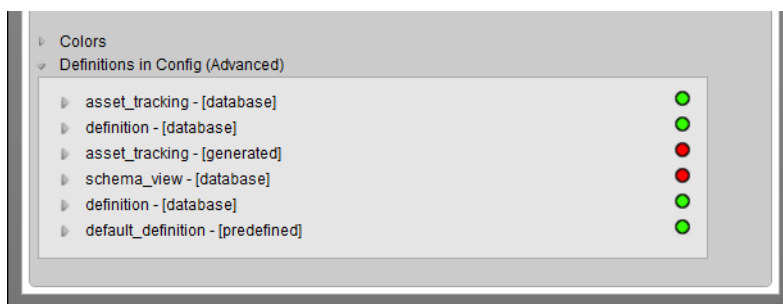
**Save to Current View:** Occasionally, it is desirable to save a view definition in the current view. This means it will not be available to other views, so this option should only be used if it is completely specific to this view.

The next 3 are the standard Undo/Redo/Show Server Transaction Log menu options for convenience.

## Widget Sections

The Element Definition Widget is broken into sections:

1. **Attributes:** These are generic attributes to describe the overall drawing of this element. All elements possess these same attributes:
  - **Title:** The title to be displayed in the column header for this widget. If it is empty, then TACTIC will use the element **Name** for the title
  - **Name:** The name of the column in the database (autogenerated based on the title when creating a new column)
  - **Width:** The default width of the column
  - **Enable Colors:** Enables the cell colors as set under the Colors section.
2. **Display:** The display section defines the configuration of the widget that will be used to display data.
3. **Colors:** Set the color for the cell for specific cell values.
4. **Definitions in Config (Advanced):** This is an advanced display and typically used to find out where in the config hierarchy a particular element definition is located.

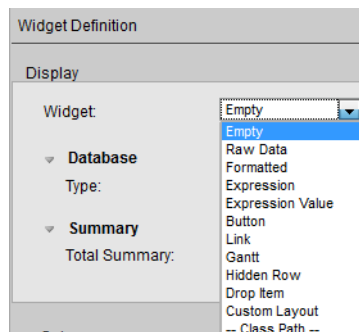


For those familiar with the widget config table, the underlying drawing mechanism does not change. XML defined widgets still drive the drawing engine, however, the Element Definition widget makes it much easier to create new elements and edit existing ones.

## Display Section

Each element has uses a drawing widget which will determine how an element will draw itself. Each drawing widget contains a number of configuration attributes which alter the functionality of the widget. The attributes displayed depend on which widget is selected.

The following widgets are predefined in TACTIC and can be selected in the drop down.



<b>Empty</b>	Specifies that no widget is to be used.
<b>Raw Data</b>	Displays the data "as is" from the database.
<b>Formatted</b>	Formats the display of the data. eg. -(\$1,234.00)
<b>Expression</b>	Use a relative TACTIC Expression to calculate what to display for each item. One expression defines what to display for the entire column.
<b>Expression Value</b>	Allow each item to be able to have an absolute TACTIC Expression to calculate what to display. If there is an expression specified, the resulting value will be displayed in the cell.
<b>Button</b>	Display an icon button that runs a JavaScript action when clicked.
<b>Link</b>	Create a hyperlink button.
<b>Gantt</b>	Horizontal bar graph for dates.
<b>Hidden Row</b>	Toggle button to open a hidden row where another widget (element) is displayed. For example a Table, Custom Layout, Edit Panel etc.
<b>Drop Item</b>	Column where items from another view can be dropped.
<b>Custom Layout</b>	Use HTML to specify what to display. Supports TACTIC Expressions and MAKO.
<b>-- Class Path --</b>	The path to a fully qualified Python class for a custom widget.

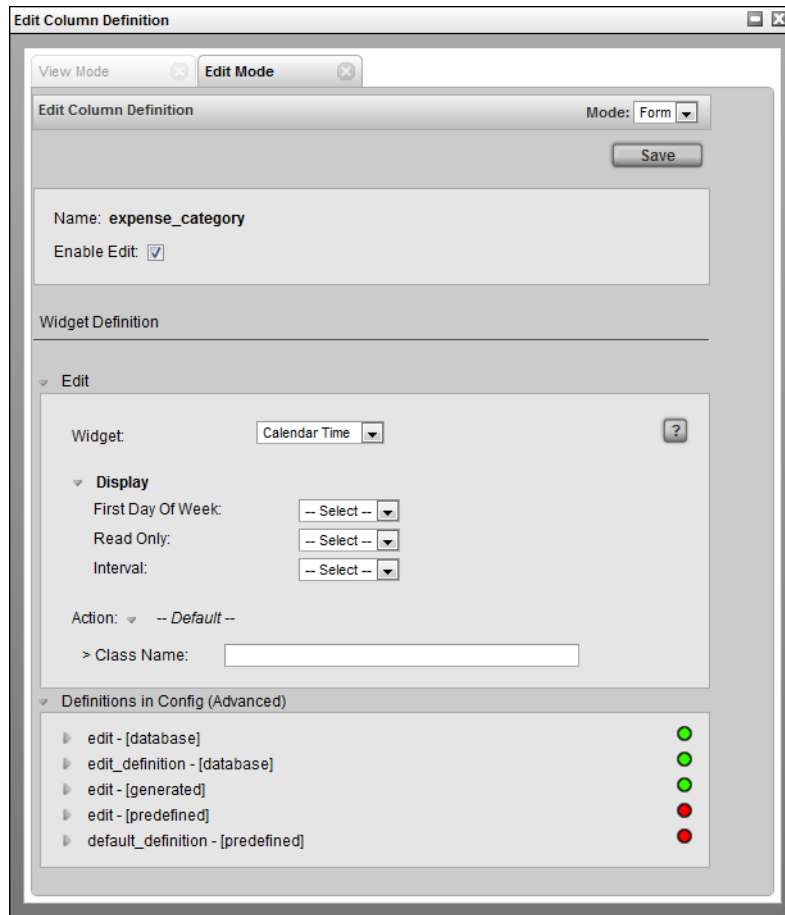
Full descriptions of each widget can be found in the widget documentation.

When "-- Class Path --" is selected, the input field to specify the path to a fully qualified python class appears. Arbitrary python classes can be specified to create complete custom widgets that are seamlessly integrated into TACTIC. Refer to the developer section below for details on how TACTIC creates the widget element interface.

When a widget or class path is selected, the available configuration attributes will be dynamically loaded.

Each widget will define its own attributes that will configure what gets displayed.

## Edit Mode Tab

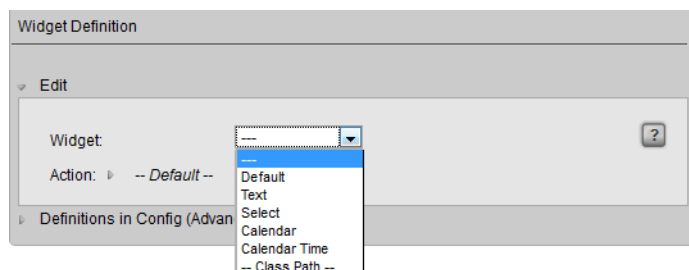


The edit section configures the edit-ability of this widget. It works very similarly to the display section except that the options are specific to edit inputs.

## Edit Section

The widgets available here are:

**Figure 1. Edit Mode Tab for the Element Definition Widget**



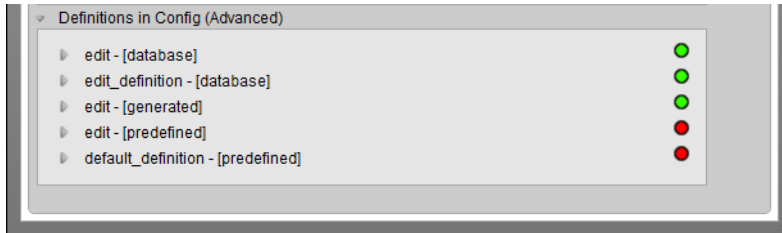
---	Specifies that no widget is to be used.
<b>Default</b>	Use the default input widget.
<b>Text</b>	Use the Text Widget as the input widget.



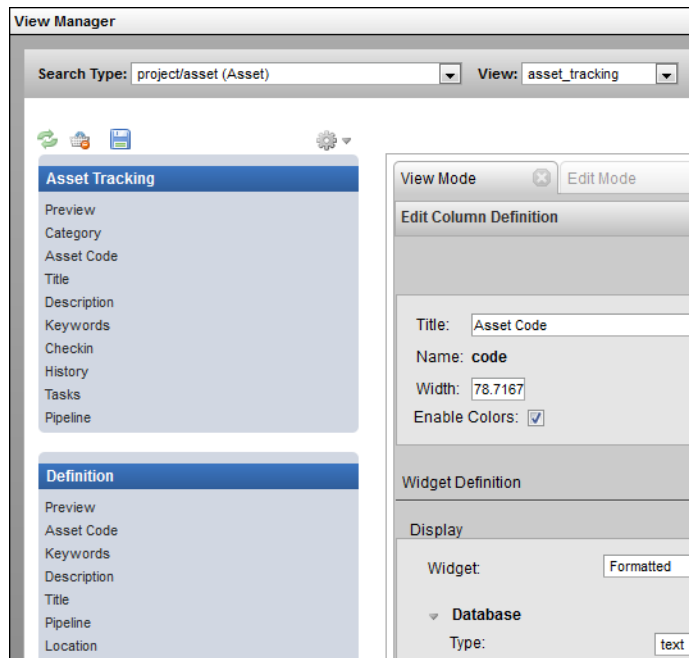
<b>Select</b>	Create a drop down selection menu and specify the selection options as the input widget.
<b>Calendar</b>	Use the Calendar Input Widget
<b>Calendar Time</b>	Use the Calendar and Time Input Widget
<b>-- Class Path --</b>	The path to a fully qualified Python class to a custom input widget.

## Advanced

This section displays the various definitions of this widget.



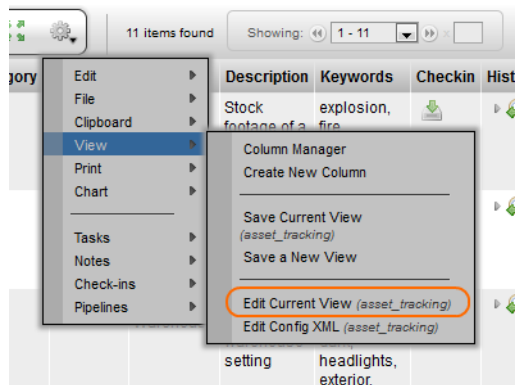
## View Manager



## Description

The View Manager provides the ability to create, edit and modify views.

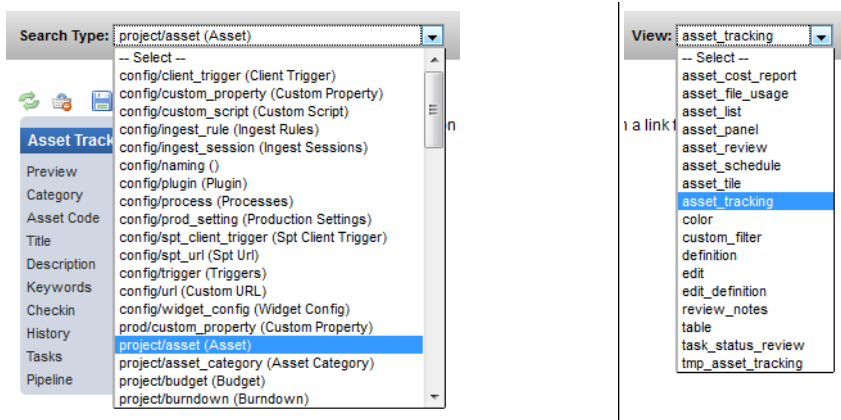
This tool can be opened to edit the current view under Gear Menu under: **View -> Edit Current View**



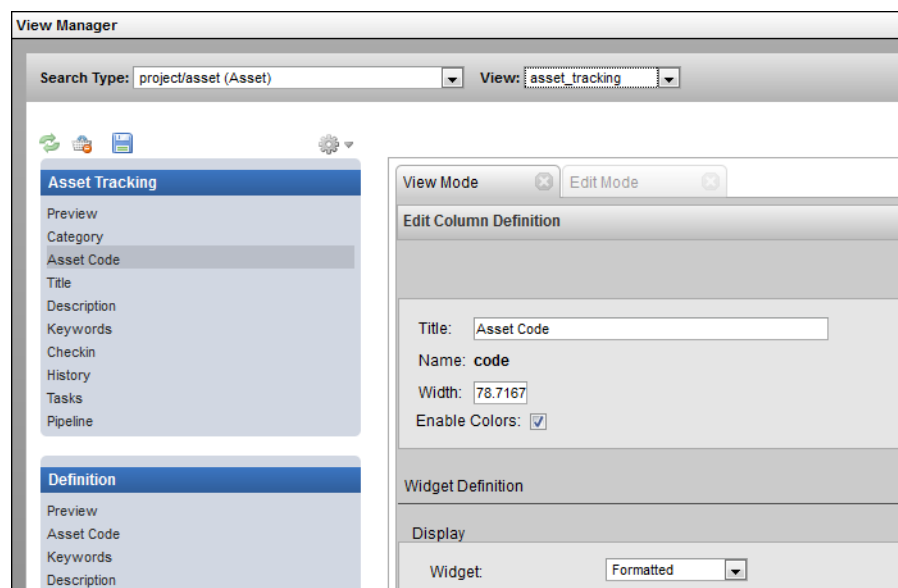
The tool can also be opened and will prompt to select a Search Type and View to open in the sidebar under: **Admin Views -> Project Admin -> Mange Config Views**.

## Implementation

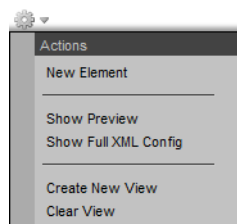
In the View Manager, select the sType and View if not already selected for the current view. The drop down selection list provides access to quickly navigate through all the views available for the selected sType.



On the side panel on the left, select an element to open the Column Definition view. The properties for the column are displayed and can be modified.

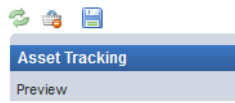


The gear menu in the View Manager provides the following options:



- **New Element** - Create and add a new custom element (column).
- **Show Preview** - Open a quick preview window of the current view.
- **Show Full XML Config** - Open an XML view of the current view in a new window.
- **Create New View** - Create a new view from the UI.
- **Clear View** - Remove all the elements from the view. (*not definition*)

The **Refresh**, **Trash** and **Save** option buttons located to the left of the gear menu.



# Users Management

## Insert a New User


The screenshot shows the 'Users' management interface. At the top, there's a toolbar with icons for refresh, add, delete, and settings. Below the toolbar, a table lists 7 users. The user 'bobby' is selected, and a modal form titled 'Edit: sthpw/login' is open. The modal form has a 'Save changes to Users (bobby)' header and contains fields for user details and login information.

Preview	Login	First name	Last name	Email
	admin	Admin		dan@southpawtech.com
	beth	Beth	Content Creator	dan@southpawtech.com
	bobby	Bobby	Supervisor	dan@southpawtech.com
	brad			
	cindy			
	mark			
	ted			

**Edit: sthpw/login**

Save changes to Users (bobby)

  
login\_bobby\_web\_icon\_v001.jpg

Preview

Login

Password

Password re-enter

First name

Last name

Email

Project code

License type

### Users and Groups

Login access to TACTIC is controlled by a user login system. In TACTIC, users can also be assigned to groups, which are used to apply various access rules.

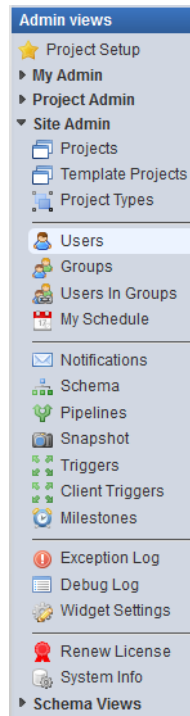


### Note

ActiveDirectory/LDAP can be used as the authentication method. Please refer to the index for those instructions.

User logins are tracked, as well as what transactions they executed. This information allows for accountability throughout the system for all users (i.e. "who did what and when").

To manage and insert users, open the **Users** view under Admin View->Site Admin->Users.



This view shows the list of all TACTIC users in the system. You must add users here for TACTIC to recognize them. To insert a new user, click on the Insert button and fill out the appropriate fields.

	Preview	Login	First name	Last name	Email
<input type="checkbox"/>		admin	Admin	Administrator	dan@southpawtech.com
<input type="checkbox"/>		beth	Beth	Content Creator	dan@southpawtech.com
<input type="checkbox"/>		bobby	Bobby	Supervisor	dan@southpawtech.com
<input type="checkbox"/>		brad	Brad	Content Creator	dan@southpawtech.com
<input type="checkbox"/>		cindy	Cindy	Client	dan@southpawtech.com
<input type="checkbox"/>		mark	Mark	Content Creator	dan@southpawtech.com
<input type="checkbox"/>		ted	Ted	Executive	dan@southpawtech.com

To edit an existing user, right click on the row and select **Edit** from the context menu. The user's password can be set here.

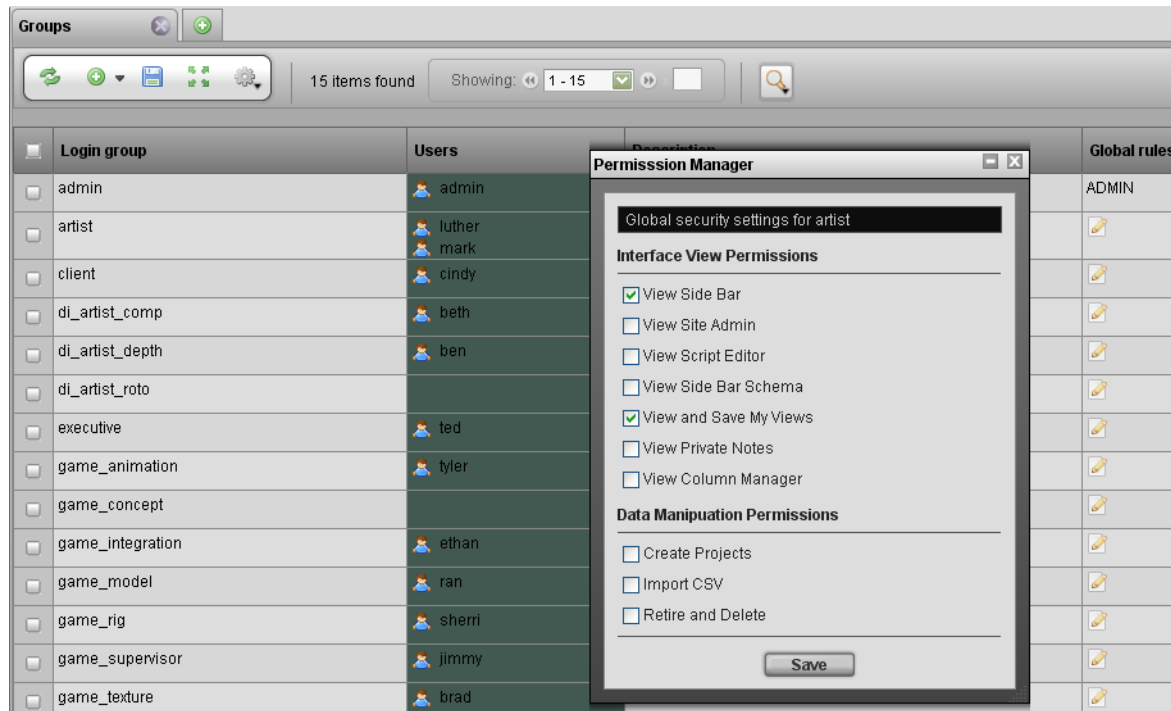
# Managing Access Rules

## TACTIC Security

All information in TACTIC goes through a series of security checks that are built into the lowest level of the software. The security architecture is a rules-based system where an access request to any piece of information must satisfy all the rules before the user gains access to it.

Each user has a login. User logins are assigned to groups, and each group can have a number of access rules attached to it. These rules determine what a user is permitted to see and do in TACTIC. At the base level, these permissions are XML structured rules that are stored in the "access\_rules" property of a group SObject. Although inserting these rules directly into the XML code allows for the most flexibility for the project manager, there are various other aspects of the TACTIC interface that can also assist in the rule creation process.

## Managing Rules



The "groups" search type contains a property (available in the column manager) named "Global rules." When this property is included in the view, a click-able button is available to load the global rules pop-up. This pop-up provides several predefined global access rules that can be applied to the group:

<b>View Side Bar</b>	View access for the complete side bar.
<b>View Site Admin</b>	Allow access to see the "Site Admin" section of the side bar.
<b>View Script Editor</b>	Access to the Script Editor
<b>View Side Bar Schema</b>	Allow access to the schema section of the side bar.
<b>View and Save My Views</b>	Save personal My Views
<b>View Private Notes</b>	Allow viewing of private notes.
<b>View Column Manager</b>	Allow viewing of the column manager

<b>Create Projects</b>	Allow creating of new projects.
<b>Import CSV</b>	Import CSV Files.
<b>Retire and Delete</b>	Allow the ability to retire and delete in the right-click context menu..

To customize the options for these rules, click the edit icon in the Global Permissions column for the desired group. From the rule selection pop-up that appears, select one of the options. When you click the save button, they are committed to the access rule XML for the chosen group.

## Side bar Manager Security

You can select which groups can see each of the links in the TACTIC side bar manager.

Group	Group default
<input checked="" type="checkbox"/> admin	allow all
<input checked="" type="checkbox"/> artist	allow all
<input type="checkbox"/> client	allow all
<input type="checkbox"/> di_artist_comp	allow all
<input type="checkbox"/> di_artist_depth	allow all
<input type="checkbox"/> di_artist_roto	allow all
<input type="checkbox"/> executive	allow all
<input type="checkbox"/> supervisor	allow all

The Element Detail window lists all groups in the system. Check any group to allow access (or uncheck to deny access). When you click the **Save Definition** button, your changes are saved to each group's "access\_rules" property. To view your changes in the XML code for any of the groups, navigate to a group view which has the "access\_rules" property column.



# Project Automation

## TACTIC Event System Introduction

The TACTIC Event System is built into the base transactional system in Tactic's core. Every transaction which occurs in Tactic can fire an event which in turn, can be used to execute a trigger or notification.

These events can be incorporated to automate specific processes that are often repetitive. At the simplest level, there are interfaces that help prepare and configure these aspects but, it is good to understand how they work. Overall, there are 2 levels that these events can be configured. The first is using the predefined event options provided in the Project Workflow or Project Schema interfaces and the second in the low level database events.

### Predefined Events

The following list of events are the events provided in the Project Workflow interface. For more information in setting up Notifications and Triggers with this interface, please refer to **Project Automation - Triggers** and **Project Automation - Notifications**

<b>A task Status is Changed</b>	When the status of a task is changed. Further options are provided allowing for selection.
<b>A new note is added</b>	When a new note (sthpw/note) is added to the project.
<b>A task is assigned</b>	When a task is assigned to a user.
<b>Files are checked in</b>	When files are checked in to an SObject.
<b>Files are checked out</b>	When files are checked out from an SObject.
<b>Custom event</b>	Allows for calling of an event using the raw Database Events.

### Raw Database Events

Below is the list of the database level events. These events are run regardless of how they are called (interface, api, external integration etc)

<b>done</b>	Executed each time a transaction completes
<b>insert</b>	Executed each time a Search Object has been inserted.
<b>update</b>	Executed each time a Search Object has been updated.
<b>change</b>	Executed each time a Search Object has changed. This combines the events insert, update and delete.
<b>retire</b>	Executed each time a Search Object has been retired.
<b>delete</b>	Executed each time a Search Object has been deleted.
<b>checkin</b>	Executed each time a checkin occurs for a Search Object
<b>checkout</b>	Executed each time a checkout occurs for a Search Object
<b>timed</b>	Executed on a timed interval. This is only supported for triggers.

For example, in a transaction where the status of a task is being changed, an association to this event can be made with the following notation:

```
update|sthpw/task|assigned
```

The notation can consist of 3 sections although only the event is required.

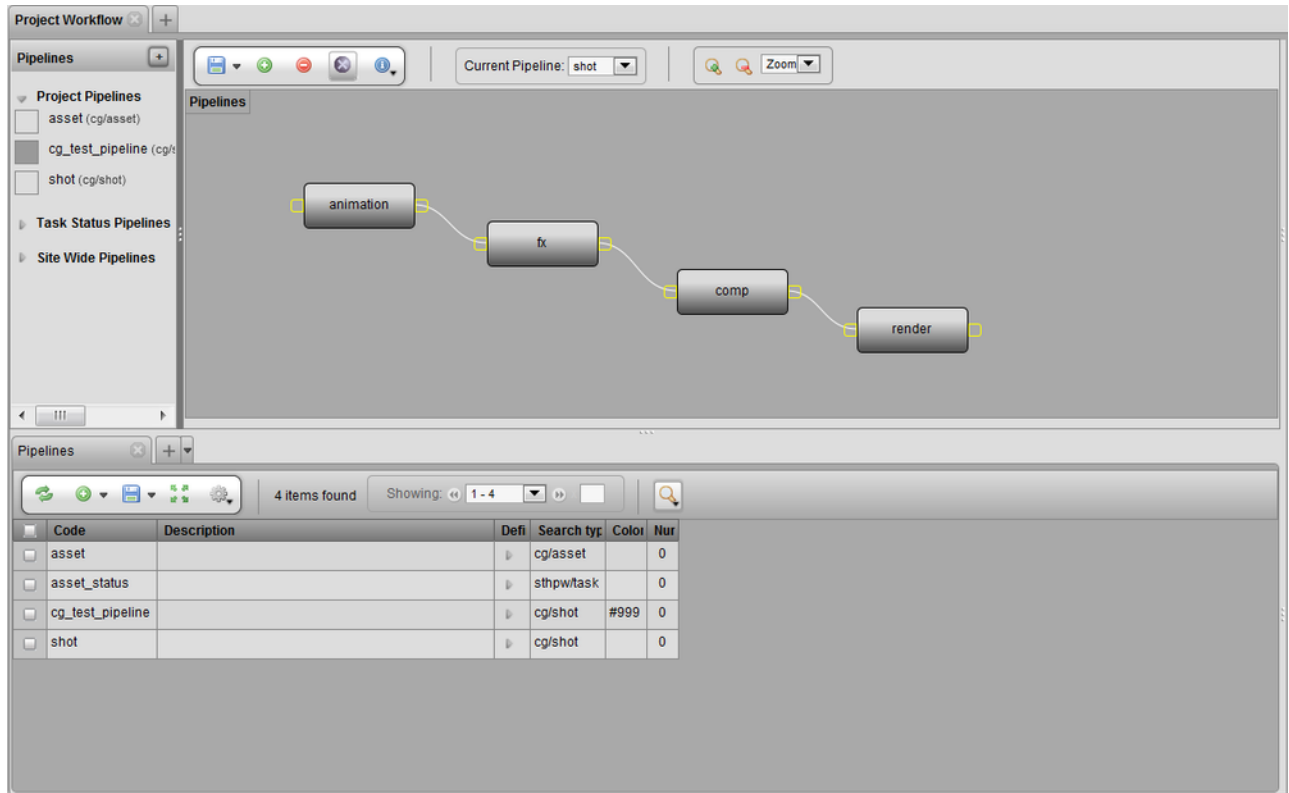
<Event> | <SType> | <Column>

<b>Event</b>	The raw database event.
<b>SType</b>	The Searchable Type (SType) the event is occurring for.
<b>Column</b>	The Column that was changed in the SType.

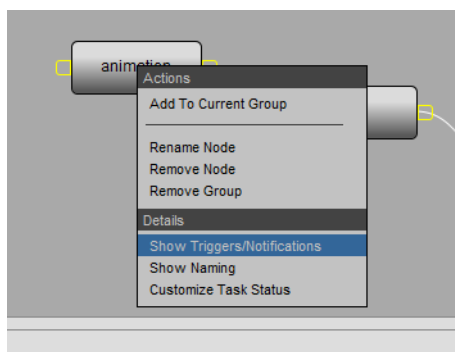
## Project Automation - Triggers

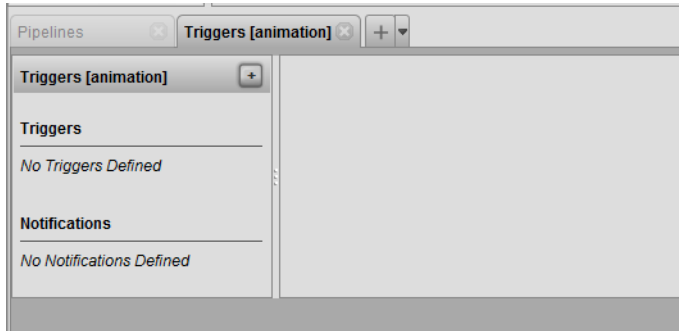
Triggers are events that are called upon a transaction to automate workflow. These triggers can be accessed within the Project Workflow view.

**Admin Viwes -> Project Admin -> Project Workflow**

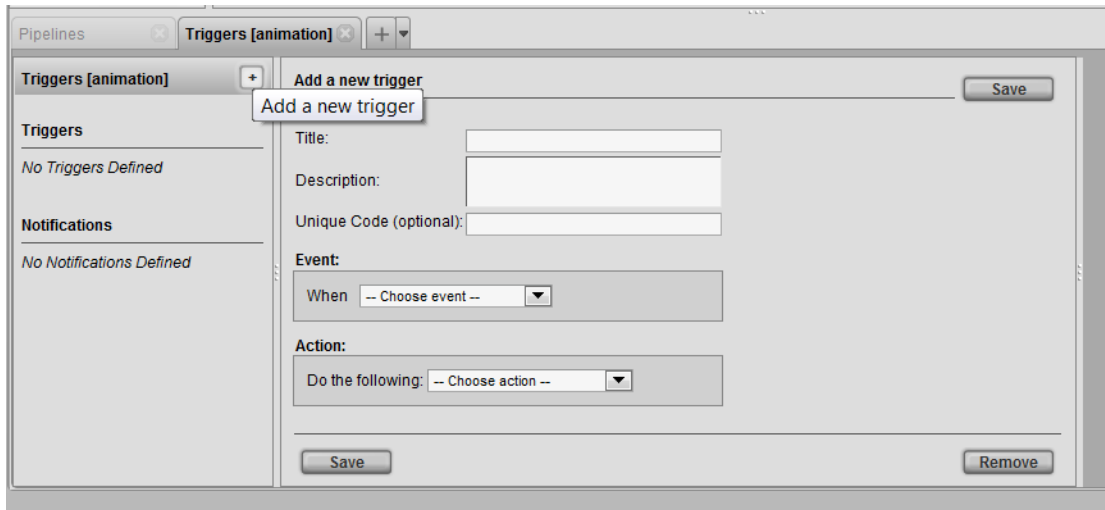


Each process in the pipeline can have their own triggers. Right clicking on a process and choosing **show notification/trigger** option will open a tab to define a trigger for that specific process.





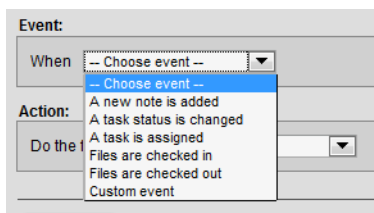
The trigger tab will also display the assigned process. Clicking the insert button will open the trigger UI.



<b>Title</b>	Title of the trigger.
<b>Description</b>	Description of the trigger.
<b>Unique Code</b>	
<b>Event</b>	Drop down list of trigger events. This event is where the trigger is called.
<b>Action</b>	The action is what the event will

## EVENT

The Events drop down list provides a wide range of different triggers.



Depending on the trigger the Event box may show additional options.

**A new note is added** - This Event will be called when a new note is inserted into the process.

**A task status is changed** - This event will be called when a status is changed. The event box also gives a additional option to choose specific status.

The screenshot shows an 'Event:' configuration box. It has a 'When' dropdown menu set to 'A task status is changed'. Below it, there is a 'to:' dropdown menu set to 'Approved'.

**A task is assigned** - This event will be called when any task in the specified process is assigned.

**Files are checked in** - This event will be called when there is a checkin to the specified trigger. This event also gives a additional option to choose what process the action will effect.

The screenshot shows an 'Event:' configuration box. It has a 'When' dropdown menu set to 'Files are checked in'. Below it, there is a section 'In the following process:' with a checkbox labeled 'this process' (which is checked) and a dropdown menu set to '-- Previous Process --'.

**Files are checked out** - This event will be called when there is a checkout to the specified trigger.

**Cusom Event** -

The screenshot shows an 'Event:' configuration box. It has a 'When' dropdown menu set to 'Custom event'. Below it, there is a text input field labeled 'Event name:'.

## ACTION

The Action drop down list provides a series of predefined actions that work with the above events.

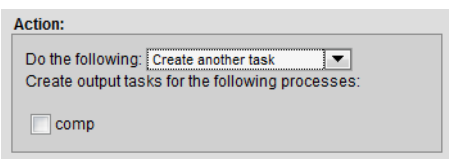
The screenshot shows an 'Action:' configuration box. It has a 'Do the following:' dropdown menu. The dropdown menu is open, showing options: '-- Choose action --', 'Send a notification', 'Update another task status', 'Create another task', 'Run python code', and 'Run python trigger'. A 'Save' button is visible below the dropdown.

**Send a notification** - See project automation notification docs.

**Update another task status** - This action will update a task status. This action also opens additional options to update status of current and other process of the pipeline as well as the option of status.

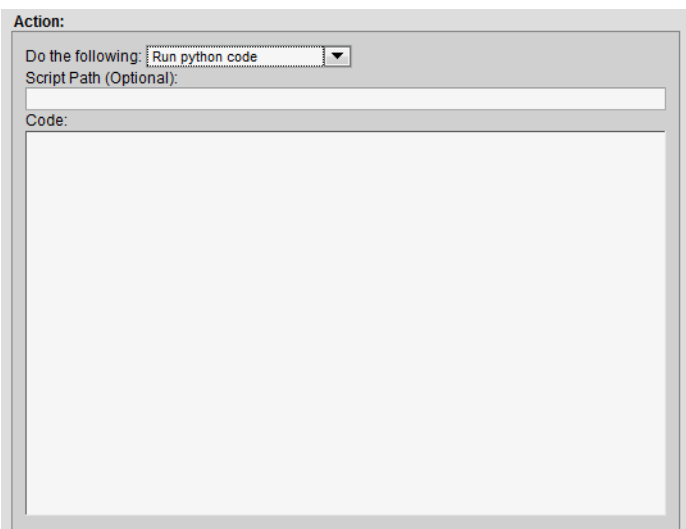
The screenshot shows an 'Action:' configuration box. It has a 'Do the following:' dropdown menu set to 'Update another task status'. Below it, there are three checkboxes, each followed by a status dropdown menu set to 'Assignment':  
☐ Set "tx" status to Assignment  
☐ Set "animation" status to Assignment  
☐ Set "comp" status to Assignment

**Create another task** - This Action will create a task upon event. This action also opens additional options to choose from creating a task in current or next process.



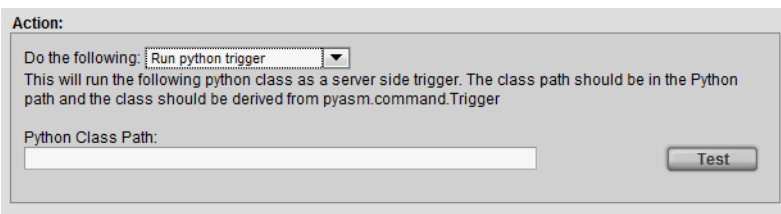
The screenshot shows a configuration window titled "Action:". Inside, there is a section "Do the following:" with a dropdown menu set to "Create another task". Below this is the text "Create output tasks for the following processes:". At the bottom, there is a checkbox labeled "comp" which is currently unchecked.

**Run python code** - This Action will run python code upon event. The action box opens additional options to name and insert a python code.



The screenshot shows a configuration window titled "Action:". Inside, there is a section "Do the following:" with a dropdown menu set to "Run python code". Below this is a text field labeled "Script Path (Optional):". At the bottom, there is a large text area labeled "Code:" for entering the python code.

**Run python trigger** - This Action will run python trigger upon event. The action box opens additional option to insert the name of the trigger. These can be custom written scripts that can be called from Tactic's API.



The screenshot shows a configuration window titled "Action:". Inside, there is a section "Do the following:" with a dropdown menu set to "Run python trigger". Below this is a text area containing the instruction: "This will run the following python class as a server side trigger. The class path should be in the Python path and the class should be derived from pyasm.command.Trigger". At the bottom, there is a text field labeled "Python Class Path:" and a "Test" button.

When satisfied with the options set to run a trigger, the trigger must be saved in order to be applied. When the trigger is saved the title of the trigger will appear beneath the triggers panel.

The screenshot displays the TACTIC Setup interface. On the left, a sidebar contains a 'Triggers [animation]' section with a '+' icon, a 'Triggers' list showing 'My First Trigger', and a 'Notifications' section with the text 'No Notifications Defined'. The main area is titled 'Edit existing trigger' and contains the following fields:

- Trigger:** 3CG - This trigger will update next process status.
- Title:** My First Trigger
- Description:** This trigger will update next process status.
- Unique Code (optional):** 3CG
- Event:**
  - When:** A task status is changed
  - to:** Approved
- Action:**
  - Do the following:** Update another task status
  - ☐ Set "animation" status to Assignment
  - ☐ Set "fx" status to Assignment

A 'Save' button is located at the bottom of the main area.

There are no limitations of how many triggers you can have. Each process can have multiple triggers applied.

# Project Automation - Notifications

## Description

Notifications are sent to inform the user that a particular transaction or event has occurred.

They are stored in a notification\_log which can be found under **Admin Views -> Site Admin -> Notifications**.

Notifications present information reported by transactions. They usually include what items are created or updated in addition to a description of the command. Below is an example of a notification:

	Project code	Login	Command cls	Subject	Message	Timestamp
<input type="checkbox"/>	support	admin		TACTIC Ticket 49 Has been updated.	southpaw TACTIC Ticket 49 has been updated. Subject: I want to know how to plant trees Summary: Do I need a shovel or a rake? Status: open Please do not reply to this e-mail. To reply or to make further comments, please login to the Ticketing system @ <a href="http://tickets.southpawtech.com/tactic">http://tickets.southpawtech.com/tactic</a>	Sep 21, 2009
<input type="checkbox"/>	support	admin		TACTIC Ticket 68	admin TACTIC Ticket 68 has been updated.	Sep 21, 2009

With the mail server setting set up properly (set in the TACTIC Config file), TACTIC can send out email notifications to users.

<div> <div> <div></div> <div>Subject: TACTIC Ticket 49 Has been updated.</div> </div> <div> <div>From: <a href="#">Southpaw Support</a></div> <div>Reply-To: <a href="#">Southpaw Support</a></div> <div>Date: 4:36 PM</div> <div>To: <a href="#">Southpaw Support</a>, <a href="#">Support Moderator</a>, <a href="#">Boris Lai</a></div> </div> </div>
<div> <div>southpaw TACTIC Ticket 49 has been updated.</div> <div>Subject: I want to know how to plant trees</div> <div>Summary: Do I need a shovel or a rake?</div> <div>Status: open</div> <div>Please do not reply to this e-mail. To reply or to make further comments, please login to the Ticketing system @ <a href="http://tickets.southpawtech.com/tactic">http://tickets.southpawtech.com/tactic</a></div> </div>

## Implementation

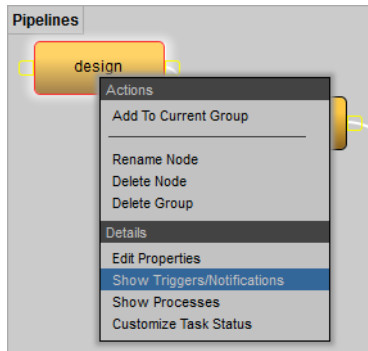
There are 2 perspectives to work from when configuring notifications in TACTIC.

- **Project Workflow** - Notifications can be set up per process in a pipeline which are used to automate the pipeline/workflow
- **Project Schema** - Notifications can be set up at a simpler level where any of the Raw Database events can be used to trigger a notification regardless of process.

### Project Workflow

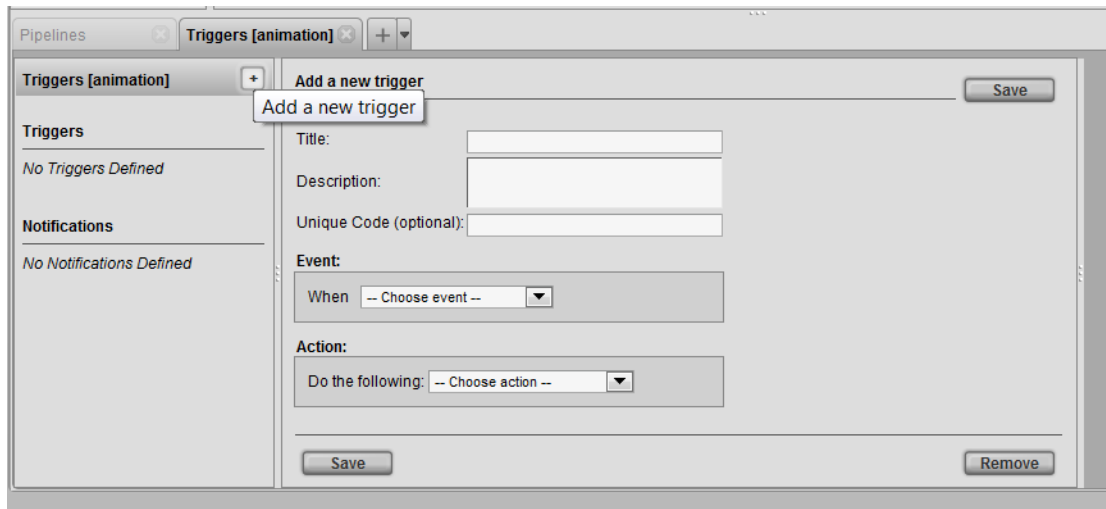
In the Workflow Editor, right click on a process and choose **show notification/trigger** to open a tab to define a trigger for that particular process.





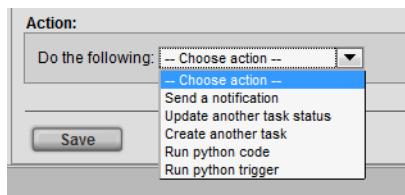
This will open a new Triggers tab in the panel at the bottom for the assigned process.

Click the [+] button to insert a trigger. This will open the trigger/notification UI.



Notifications and Triggers work together in many ways. A notification is defined as an Action. To send a notification, an event must occur.

In the Action drop down list **Send a Notification** must be selected.



**Send a Notification** - This action will send a notification. The action box will open additional options to insert a subject and message.

## Example 1

Below is an example of a notification being sent on the event when a task status is changed to review:

<b>Event:</b>	
When	A task status is changed ▼
to:	Review ▼
<b>Action:</b>	
Do the following:	Send a notification ▼
Subject:	{{@GET(sthpw/project.title)}} {@GET(parent.code)} - {@GET(process)} has been set to {@GET(status)}
Message:	<pre>{@GET(sthpw/project.title)}  {@GET(parent.code)} {@GET(process)} has been set to {@GET(status)}  Assigned: {@GET(assigned)} Due: {@FORMAT(@GET(bid_end_date), 1999-12-31)}</pre>
Mail To:	{@GET(assigned)}, supervisor
Mail CC:	

The **Mail To:** and **Mail CC:** input fields accepts the following types of input:

**Email** - Capability to add regular emails allows to send personal email addresses e.g. joe@my\_email.com

**Group** - Capability to send to a group of users in TACTIC e.g. Supervisor

**Expression** - Capability to insert expressions that specifies a user in TACTIC. All expressions are identified by curly brackets "{}". e.g. {@SUBJECT(sthpw/login)}

**Send a Notification** - This action will send a notification. The action box will open additional options to insert a subject and message.

## Example 2

Below is an example which uses more expressions for a notification being sent whenever a task is assigned.

**Event:**

When:

**Action:**

Do the following:

**Subject:**

{{@GET(sthpw/project.title)}} {{@GET(parent.code)}} - {{@GET(.process)}} has been assigned to you.

**Message:**

Hello {{@GET(sthpw/login.first\_name)}},

The following task has been assigned to you:

{{@GET(sthpw/project.title)}} {{@GET(parent.code)}} - {{@GET(.process)}}

This task is due: {{@FORMAT(@GET(.bid\_end\_date), 1999-12-31)}}

{{SERVER}}

You can view this asset at the following link:  
[## Project Schema](http://demobuild2.southpawtech.com/tactic/project/#/subject/{{@GET(.search_type)}}/{{@GET(parent.code)}}</a></p>
<p><b>Mail To:</b></p>
</div>
<div data-bbox=)

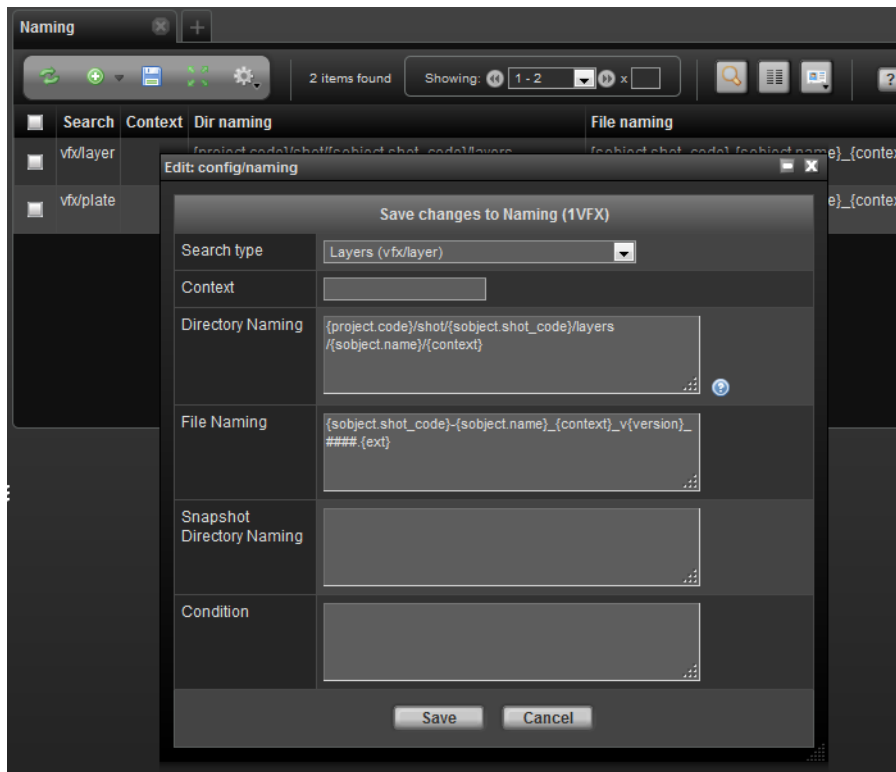
All notification configurations can be accessed through **Admin Views -> Site Admin-> Notifications**

<input type="checkbox"/>	Email test	Event	Subject	Message	Process
<input type="checkbox"/>	<input type="button" value="Email Test"/>	update sthpw/task status	TACTIC: {{@GET(parent.code)}} {{@GET(.context)}} Updated to {{@GET(.status)}}	<div> {{@GET(parent.code)}} has been updated. </div> <hr/> <div> Task: {{@GET(.context)}}  Assigned: {{@GET(.assigned)}}  Status: {{@GET(.status)}} </div>	

If there is no process specified, then the notification will be triggered regardless of process. e.g. during a snapshot or a checkin

# Project Automation - File Naming

## Introduction



The naming page provides a way of controlling directory and file naming conventions through a simple expression language. Each naming entry can contain a directory naming and/or file naming. It is designed so that a relatively non-technical user can create custom naming conventions for various sTypes.

The relative path expression language is a simple language, but in order to understand it you must know the basic components that generally make up a naming convention. The expression language allows access to almost any data in TACTIC. The keywords which are the most relevant in naming conventions are as follows:

<b>parent:</b>	The parent sObject to the current sObject defined by the search type attribute
<b>sObject:</b>	The actual sObject which is being referenced
<b>snapshot:</b>	The child snapshot generated or being referenced for the naming convention. This contains the context and version information.
<b>file:</b>	The file object pertinent to the check-in. This allows for reference to the original file name or extension.
<b>login:</b>	The user who is carrying out the check-in or check-out.
<b>user:</b>	The user who is carrying out the check-in or check-out.

The properties of these Search Objects are used to build up the naming convention.

A simple example of a relative path is as follows:

```
{project.code}/sequence/{parent.code}/{subject.code}/{snapshot.context}/maya
```

which after translation could be translated into:

```
sample3d/sequence/XG/XG001/model/maya
```

This expression is explicit in that every variable scopes the object that the attribute comes from.

Another example is for a file name:

```
{subject.code}_{snapshot.context}_v{snapshot.version}.{ext}
```

This can be translated into: chr001\_model\_v001.ma upon expansion.

The @GET(sobejct.code) or @GET(subject.sthpw/snapshot.version) TACTIC expression language syntax can be used instead; however, the original keyword approach is more readable. In case you do decide to use the TEL syntax, here are the equivalents:

```
{basefile} = {$BASEFILE}
```

```
{ext} = {$EXT}
```

```
{project.code} = {$PROJECT}
```

```
{login} = {$LOGIN}
```

It is important to note that you can't fix TEL syntax with the keyword syntax in the same field of a naming entry.

### 3.0 Defaults

TACTIC will fall back on the default convention which would be represented by the following expression. These defaults are slightly different from previous versions:

```
Dir Naming: {project.code}/{search_type}/{subject.code}/{context}
```

```
File Naming: {file.name}_{context}_v{version}.{ext}
```

Checking in the file **characterFile.ma** would create the following file and directory structure:

```
assets/sample3d/characters/chr001/publish/characterFile_v001.ma
```

### Assumptions

Various assumptions have been made about which attributes are attached to which SObjects. It is often the case that the context is composed of a number of parts that are of interest to a naming convention.

For example, it is conceivable to have a context named: "model/hi". However, you may wish to break this up in a specific way in your naming convention. This is accomplished using [] notation common to many programming languages.

The following notation could be used for a directory using this: which could be translated into

```
Dir Naming: {code}/{context[0]}/maya/{context[1]}
```

```
Result: chr001/model/maya/hi
```

## Inserting Naming Conventions

To insert a naming convention expression, load a **Naming** view and click the Insert button to insert a new set of expressions.

A Naming Convention sObject has specific properties which are used to either define the convention or act as conditions to define if the convention should be used for the given checkin. When Inserting, fill in the following options:

<b>Search Type</b>	The search_type to associate the naming convention to.
<b>Snapshot Type</b>	The snapshot type of the checkin to use as a condition (default for most TACTIC check-ins is 'file'. Default for directory checkin using the General Checkin Widget is 'dir'. Since this is a more advanced attribute, it is hidden by default)
<b>Context</b>	The snapshot context of the checkin to use as a condition (default checkin when there is no pipeline is 'publish')
<b>Dir Naming</b>	The expression entry for the directory naming convention
<b>File Naming</b>	The expression entry for the file naming convention
<b>Sandbox Dir Naming</b>	The expression entry for the user sandbox directory naming convention
<b>Latest versionless</b>	If set to true(checked), every time a check-in is created, a latest version of the main file will be created as well. If you want to always have a file that refers to the latest version of a model you can use this feature by calling it {subject.code}_{context}_latest.{ext}. The latest version exists as copy by default. To make it a symlink, set the project setting versionless_mode to 'symlink'. Note: If this is checked, you need to have an entry in the naming table just for this versionless case in addition to the usual one for regular check-ins.
<b>Current versionless</b>	If set to true(checked), every time a check-in is created, a current version of the main file will be created as well. If you want to always have a file that refers to the latest version of a model you can use this feature by calling it {subject.code}_{context}_latest.{ext}. The latest version exists as copy by default. To make it a symlink, set the project setting versionless_mode to 'symlink'. Note: If this is checked, you need to have an entry in the naming table just for this versionless case in addition to the usual one for regular check-ins.
<b>Manual version</b>	If set to true(checked), the incoming file name can dictate what the version of the checked-in snapshot appears as. For instance, if the incoming file name is dessert_v005.jpg, the version will appear as version 5. Another example is sun_V0030.0010.jpg. The version will appear as version 30. It tries to recognize the number immediately after the v or V in the file name. Zero or negative numbers are ignored. If such a version already exists, the check-in will throw an error
<b>Condition</b>	It can be set up so that different naming is adopted based on a particular attribute of the sObject. For instance, for the sType prod/asset, one can assign 2 naming entries. The default naming where the condition is left blank is adopted in most circumstance. The second special naming is adopted when the category attribute equals 'vehicle' by using this expression @GET(category)=='vehicle'.

## Examples

### Example A

In the following example, the file is being checked in with the general 'publish' context

**File:** *characterFile.ma*

**Checkin Context:** publish

**Desired output:** sample3d/characters/character001/character001\_publish\_v001.ma:

**Dir Naming:** {project.code}/{parent.title}/{subject.code}

**File Naming:** {subject.code}\_{snapshot.context}\_v{snapshot.version}.{ext}

### Example B

In the following example, the shot RC\_001\_001 is part of the parent sequence RC\_001 and is checked in with an 'animation' context. This will also use the short hand expressions

**File:** shotFile.ma

**Checkin context:** animation

**Desired output:** sample3d/shot/RC\_001/RC\_001\_001/scenes/RC\_001\_001\_animation\_v001.ma

**Dir Naming:** {project.code}/shot/{parent.code}/{subject.code}/scenes

**File Naming:** {subject.code}\_{snapshot.context}\_v{snapshot.version}.{ext}

### Example C

In the following example, the desired file location is at the project folder level to accommodate a cross-project library of files.

**File:** artFile.png

**Checkin context:** publish

**Desired output:** general/art001/art001\_publish\_v001.png

**Dir Naming:** general/{subject.code}

**File Naming:** {subject.code}\_{snapshot.context}\_v{snapshot.version}.{ext}

### Example D

In the following example, a context and subcontext are used for checking in to a final process for an art asset

**File:** artFileFinal.psd

**Checkin context/subcontext:** final/colourA

**Desired Output:** finals/art001/final/photoshop/colourA/art001\_final\_colourA\_v001.psd

**Dir Naming:** finals/{subject.code}/{snapshot.context[0]}/photoshop/{snapshot.context[1]}

**File Naming:**  
{subject.code}\_{snapshot.context[0]}\_{snapshot.context[1]}\_v{snapshot.version}.{ext}

**Example E**

In the following example, the shot RC\_001\_001 is part of the parent sequence RC\_001 and is checked in with an 'animation' context. Also when a user checks out the sandbox, files should be organized into a user folder.

**File:** shotFile.ma

**Checkin context:** animation

**Desired Repo output:** sample3d/shot/RC\_001/RC\_001\_001/scenes/RC\_001\_001\_animation\_v001.ma

<b>Desired</b>	<b>Sandbox</b>	<b>output:</b>	sample3d/albert/shot/RC_001/RC_001_001/scenes/RC_001_001_animation_v001.ma
----------------	----------------	----------------	--

**Dir Naming:** {project.code}/shot/{parent.code}/{subject.code}/scenes

**Sandbox Dir Naming:** {project.code}/{login.login}/shot/{parent.code}/{subject.code}/scenes

**File Naming:** {subject.code}\_{snapshot.context}\_v{snapshot.version}.{ext}

**Example F**

In the following example, the shot RC\_001\_001 is part of the parent sequence RC\_001 and is checked in with an 'animation' context. Also when a user checks out the sandbox, files should be organized into a user folder. Since we also want to define a latest versionless, we need a second entry with the same information, and with the latest\_versionless check-box checked. In this example we are putting the versionless file in the same directory, but you can put it in a different one if desired.

**File:** shotFile.ma

**Checkin context:** animation

**Desired Repo output:** sample3d/shot/RC\_001/RC\_001\_001/scenes/RC\_001\_001\_animation\_v001.ma

<b>Desired</b>	<b>Repo</b>	<b>latest</b>	<b>versionless</b>	<b>output:</b>	sample3d/shot/RC_001/RC_001_001/scenes/RC_001_001_animation_latest.ma
----------------	-------------	---------------	--------------------	----------------	---

<b>Desired</b>	<b>Sandbox</b>	<b>output:</b>	sample3d/albert/shot/RC_001/RC_001_001/scenes/RC_001_001_animation_v001.ma
----------------	----------------	----------------	--

Entry 1

**Dir Naming:** {project.code}/shot/{parent.code}/{subject.code}/scenes

**Sandbox Dir Naming:** {project.code}/{login.login}/shot/{parent.code}/{subject.code}/scenes

**File Naming:** {subject.code}\_{snapshot.context}\_v{snapshot.version}.{ext}

Entry 2

**Dir Naming:** {project.code}/shot/{parent.code}/{subject.code}/scenes

**Sandbox Dir Naming:** {project.code}/{login.login}/shot/{parent.code}/{subject.code}/scenes

**File Naming:** {subject.code}\_{snapshot.context}\_latest.{ext}

**Latest Versionless:** true



**Example G**

In the following example, a texture file is checked in under shot RC\_001\_001. Also when a user checks out the sandbox, files should be organized into a user folder. Since we also want to define a current versionless, we need a second entry with the same information, and with the current\_versionless check-box checked. In this example, we are retaining the original file name

**File:** shotFile.ma

**Checkin context:** animation

**Desired Repo output:** sample3d/shot/RC\_001/RC\_001\_001/texture/my\_tree\_texture.jpg

**Desired      Repo      latest      versionless      output:**      sample3d/shot/RC\_001/RC\_001\_001/texture/my\_tree\_texture\_CURRENT.jpg

**Desired Sandbox output:** sample3d/albert/shot/RC\_001/RC\_001\_001/scenes/my\_tree\_texture.jpg

**Entry 1**

**Dir Naming:** {project.code}/shot/{parent.code}/{subject.code}/textures

**Sandbox Dir Naming:** {project.code}/{login.login}/shot/{parent.code}/{subject.code}/textures

**File Naming:** {basefile}.{ext}

**Entry 2**

**Dir Naming:** {project.code}/shot/{parent.code}/{subject.code}/textures

**Sandbox Dir Naming:** {project.code}/{login.login}/shot/{parent.code}/{subject.code}/textures

**File Naming:** {basefile}\_CURRENT.{ext}

**Current Versionless:** true

**Example H**

In the following example, asset file name is made up of asset\_code, context, and version by default. If the asset's category is '2D', we will add this category as a suffix to the name

**File:** my\_chr001.jpg

**Checkin context:** model

**Desired Repo output:** sample3d/asset/chr001/chr001\_model\_v001.jpg

**Second Desired Repo output:** sample3d/asset/chr003/chr003\_model\_v001\_2D.jpg

**Desired Sandbox output:** sample3d/dan/asset/chr001/my\_chr001.jpg

Entry 1

**Dir Naming:** {project.code}/asset/{subject.code}

**Sandbox Dir Naming:** {project.code}/{login.login}/asset/{subject.code}

**File Naming:** {subject.code}\_{context}\_{version}.{ext}

Entry 2

**Dir Naming:** {project.code}/asset/{subject.code}

**Sandbox Dir Naming:** {project.code}/{login.login}/asset/{subject.code}

**File Naming:** {subject.code}\_{context}\_{version}\_{subject.category}.{ext}

**Condition:** @GET(.category)=='2D'

**Example I - keywords: snapshot and file**

The following is an example of the proper way to use the special keywords **snapshot** and **file** in an expression to retrieve the snapshot and file object for a check-in:

```
{@GET(snapshot.context)}
```

```
{@GET(file.type)}
```

Notice that the syntax for these particular expressions deviates from the syntax of typical TACTIC expressions.

ie. after the "@GET", the typical prefix: "{project.code}/{search\_type}" has been eliminated, since it not possible to filter down to the exact snapshot through the @GET(sthpw/snapshot[]) approach.

These expressions can be helpful when multiple items are checked in on a single TACTIC transaction.

Below is an example using these expressions of a file being checked in with the general 'publish' context:

**File:** source\_art\_v0001.jpg

**Checkin Context:** publish

**Desired output:** sample3d/chr/chr001/publish

**Dir Naming:** {\$PROJECT}/chr/{@GET(.code)}/{@GET(snapshot.context)}

**File Naming:** source\_art\_v{@GET(snapshot.context).version,%04.d}.{@GET(file.type)}



# Expression Language

## TACTIC Expression Language Introduction

### Introduction

This document describes the construct of the **TEL** TACTIC Expression Language. This language is a shorthand form to quickly retrieve information about related Search Objects. The expression either starts with a list of Search Objects and the operations of the expression language operate on these lists (this is quite similar to LISP in concept) or it can be used as an absolute search.

The expression language also borrows from spreadsheet syntax which is familiar to many people. The reason behind using an expression language is that it is much simpler and compact than using code or direct SQL. The TACTIC expression language is designed to be able to easily retrieve data in a single command that would otherwise take many lines of code.

### Simple Example

The expression often starts with a list of Search Objects and then operates on these Search Objects.

If you have a list of "prod/shot" Search Objects, then the following:

```
@GET(prod/shot.code)
```

will return a list of codes of these Search Objects. The notation for the method GET is of the form <search\_type>.<column>. As will be shown below, multiple search\_types can be strung together to navigate across related search\_types. The @GET function operates on a list and returns a list.

If no starting Search Object is given, then this expression will return a list of codes for every shot in the project.

### Searching

The expression language can be used as a shorthand for search for Search Objects. This is often convenient because the expression language is a pure string and can be stored a number of formats, including XML.

The @SOBJECT method will retrieve entire Search Objects.

Search for all assets

```
@SOBJECT(prod/asset)
```

Search only for characters by applying a filter

```
@SOBJECT(prod/asset['asset_library','chr'])
```

You can also apply multiple filters. AND operation is implied

```
@SOBJECT(prod/asset['asset_library','chr']['timestamp','>','2009-01-01'])
```

You can also apply multiple filters. To use OR operation with more than 2 filters. For example, with code containing the word prop1, OR asset\_library is chr, OR timestamp after 2009-01-01. Note: EQ stands for case-sensitive match.

```
@SOBJECT(prod/asset['begin']['asset_library','chr']['timestamp','>','2009-01-01']  
['code','EQ','prop1']['or'])
```

To use OR operation with 2 filters followed by an AND operation. For example, with asset\_library is chr OR timestamp after 2009-01-01 AND code containing the word prop1. If there are only 2 filters, there is no need to sandwich it with begin.

```
@SUBJECT(prod/asset['asset_library','chr']['timestamp','>','2009-01-01']['or']  
['code','EQ','prop1'])
```

Note that full filter operations from the Client API are supported.

## Navigating Search Types

One of the true powers of the expression language is the simplicity in which it can navigate between various related Search Types using a navigational syntax. The expression language makes use of the project schema to navigate dependencies between the search\_types. For example a sequence is related to a shot.

The navigational syntax is used as arguments for many aggregate methods. When detected, the expression language will perform a search through the hierarchy to retrieve the desired search results.

A simple example of the navigation syntax in the expression language is as follows:

```
@GET(prod/sequence.code)
```

This expression will get all of the codes of the sequences of related to each Search Object.

The expression can also navigate multiple levels of search types to dig deeply into the hierarchy. For example, this will get all of the descriptions of all of the episodes that belong to the sequences of the original shots.

```
@GET(prod/sequence.prod/episode.description)
```

Another useful illustration is to get all of the tasks of all of the shots:

```
@SUBJECT(prod/shot.sthpw/task)
```

## Aggregate functions

The expression language defines a number of aggregate functions which will operate on the list.

This will give the addition of all the duration attributes of the provided shots.

```
@SUM(prod/shot.duration)
```

This will give the average duration attribute of all of the shots.

```
@AVG(prod/shot.duration)
```

This will give a count of all of the Search Objects

```
@COUNT(prod/shot)
```

All of these aggregates return a single value which can be used to operate on other lists.

## Operations

The expression language operates on lists. The operator will operate on each element of the list independently and return a list. For example when doing a subtraction operation on items:

```
@GET(prod/shot.end_frame) - @GET(prod/shot.start_frame)
```

The first @GET will return a list of start frames and the second @GET will return a list end frames. When two lists are operated on the results are calculated based on items at the same position in each list. So if we had two lists:

```
[300, 155, 100] - [100, 100, 100] = [200, 55, 0]
```

Similarly, lists will be multiplied as follows

```
[5, 4, 3] * [5, 4, 3] = [25, 16, 9]
```

The expression language supports most operation support by the python language itself.

```
>>> Search.eval("5 * 25")
125.0

>>> Search.eval("5 + 25")
30.0

>>> Search.eval("(25 - 5) * 5")
100.0

>>> Search.eval("5 / 25") 0.20000000000000001

>>> Search.eval("@COUNT(sthpw/task) * 5")
2310.0

>>> Search.eval("@COUNT(sthpw/task) > 0")
True

>>> Search.eval("@COUNT(sthpw/task) == 462")
True

>>> Search.eval("@COUNT(sthpw/task) != 462")
False
```

The expression language also supports the regular expression syntax

The following tests whether the name\_first column starts with "John"

```
@GET(.name_first) ~ '^John'
```

## More complex operations

You can do more complex operations by combining the above. The following will return a cost list of all of the shots (assigned user wage \* number of hours worked).

```
@GET(prod/shot.sthpw/task.sthpw/login.wage) * @GET(prod/shot.num_hours)
```

You could add them all together using @SUM this to get the total

```
@SUM(
  @GET(prod/shot.sthpw/task.sthpw/login.wage) * @GET(prod/shot.num_hours)
)
```

There are times the sObjects returned are not unique. The @UNIQUE operator can be used to return a unique list of result. The following returns the unique list of login sObjects related to the task list provided. The @COUNT operator computes the total number of login sObjects.

```
# my.tasks is a list of tasks
expression = "@COUNT(@UNIQUE(@SOBJECT(sthpw/login)))"
result = my.parser.eval(expression, my.tasks)
```

## Manipulating Strings

Most of the operations in the expression language operate on lists and either return lists or return single values. However, it is often required that expressions be used in string concatenation. A simplified notation is to use curly brackets {} to represent an operation that converts the result of an expression into a string.

For a file to be named chr001\_model.png, we could use:

```
{ @GET(prod/asset.code) } _ { @GET(sthbw/snapshot.context) } .png
```

\*\* The file naming conventions do not current use the expression language. The presently use a simplified expression language. The plan is to merge the two at some point.

## String Formatting

For string values, the string operator them can use standard print formatting:

```
v{ @GET(sthbw/snapshot.version), %0.3d }
```

will return "v012", for example.

The expression language also supports formatting through regular expressions

```
{ @GET(prod/asset.description), | ^(\w{5}) | }
```

This will get the first 5 word characters for the description. Since the full expression language is supported, it is possible to extract a wide variety of parts. Anything matched with () will be returned as the value.

\*\*If there are multiple groupings, the expression language will concatenate the values together.

The following will return the first 3 and last 3 characters of the description.

```
{ @GET(prod/asset.description), | ^(\w{3}).*(\w{3})$ | }
```

The following will return the last 5 characters of the description of the current SObject even if it is written in French or Chinese.

```
{ @GET(.description), | ^(.{5})$ | }
```

## Time related formatting

The following formats a timestamp by extracting just the month and date (old way):

```
{ @GET(.timestamp), %b-%m }
```

The following formats a timestamp by extracting just the year

```
{ @GET(.timestamp), %Y }
```

The following removes the hours, minutes and seconds from the built-in \$TODAY variable so only 2011-11-11 is displayed

```
{ $TODAY, | ( [ ^ \ s ] + ) | }
```

The following formats a timestamp by using the new @FORMAT function

```
@FORMAT( @GET(.timestamp), '31/12/1999' )
```

```
@FORMAT( @GET(.timestamp), 'Dec 31, 1999' )
```

Either of the following formats a frame count into timecode

```
@FORMAT( @GET(.frame_count), 'MM:SS.FF' )
```

The following formats a frame count into hours, minutes and seconds in 30fps, leaving out the frames.

```
@FORMAT( @GET(.frame_count), 'HH:MM:SS', '30' )
```

The following formats a cost column in currency format

```
@FORMAT(@GET(.cost), '-$1,234.00')
```

'31/12/99 13:37' can be used to show both date and time

### Shorthand (mostly for backwards compatibility)

```
@GET(subject.end_frame) - @GET(subject.start_frame)
```

or

```
@GET(.end_frame) - @GET(.start_frame)
```

Or replicate file naming conventions

```
{subject.code}_{snapshot.context}_v{version}.{ext}
```

In the file naming convention language, there are a number of short hand keywords:

sObject Keywords: subject, snapshot, file, parent

Attribute Keywords: context, version, ext, basefile

## Expression Method Reference

### GET

@GET( [search]:nav )

v2.5.0+

The GET method will retrieve attributes or columns from a list of SObjects. This method returns a list of the values. The first argument supports the search type navigational syntax to travel through related search types.

Get the bid start date of all of the tasks:

```
@GET(sthpw/task.bid_start_date)
```

Get the assigned user of all of the modelling tasks

```
@GET(sthpw/task['context','model'].assigned)
```

The GET function also supports short hand to get all attributes from the current SObjects. This will get the assigned column for all current SObjects

```
@GET(.assigned)
```

### SUBJECT



v2.5.0+

The **SOBJECT** method is similar to the **GET SObject** except that the entire search object is retrieved.

The following expression gets all of the completed modelling tasks.

```
@SOBJECT(sthpw/task['status','complete']['context','model'])
```

The **SOBJECT** method can also traverse thru related **STypes** if their relation has been set up in the Project Schema.

The following expression gets all the shots for sequence 'SE02' and 'SE03':

```
@SOBJECT(prod/sequence['code','in','SE02|SE03'].prod/shot)
```

Certain relationships like those between anything to notes or tasks are already pre-established.

The following expression gets all the shots that have a note starting with the word 'Hello':

```
@SOBJECT(sthpw/note['note','EQ','^Hello'].prod/shot)
```

## COUNT

v2.5.0+

The **COUNT** method will return the count of the **SObject** returned by the search specifications.

To get a count of all the tasks:

```
@COUNT(sthpw/task)
```

To get a count of all the tasks of all of the shots:

```
@COUNT(prod/shot.sthpw/task)
```

To get a count of all the modelling tasks of all of the completed shots

```
@COUNT(prod/shot['status','complete'].sthpw/task['context','model'])
```

## SUM

v2.5.0+

This method will calculate a sum of all of the values in the first argument. The first argument must conform to the navigational syntax.

**AVG**

v2.5.0+

Calculates the average of all of the values of the first argument, which must conform to the navigational syntax.

**MIN**

v2.5.0+

Returns the minimum value in a list

**MAX**

v2.5.0+

Returns the maximum value in a list

**FLOOR**

v2.5.0+

Returns the lowest integer value of a passed in value

**UNIQUE**

@UNIQUE( [expr1]:expr )

v2.5.0+

The UNIQUE method goes through a list returned from an expression and ensures that only unique elements are present. Duplicates are discarded

**UNION**

@UNION( [expr1]:expr, [expr2]:expr, ... )

v2.5.0+

The UNION method combines the union of all of the results from a number of expressions together into a single list.

Combine all the users from accounting and marketing together into one list:

```
@UNION(  
  @SOBJECT(sthpw/login[ 'dept', 'accounting' ],  
  @SOBJECT(sthpw/login[ 'dept', 'marketing' ]  
)
```

**INTERSECT**

@INTERSECT( [expr1]:expr, [expr2]:expr )

v2.5.0+

The INTERSECT method takes the intersection of all the results of expressions in the arguments.

```
@INTERSECT(  
  @GET(sthpw/login['dept','supervisor']),  
  @GET(sthpw/login['dept','director'])  
)
```

## IF

@IF( [condition]:expr, [if\_true]:expr, [if\_false]:expr )

v2.6.0+

The following example will return 'red' if the number of tasks is greater than 5 and 'green' if less than or equal to 5. These types of expressions are very useful to determine colors of various backgrounds or widgets within TACTIC.

```
@IF( @COUNT(sthpw/task) > 5, 'red', 'green' )
```

Not all of the arguments can be expressions, so the values for both is\_true and is\_false can be expressions that are evaluated:

```
@IF(  
  @COUNT(sthpw/task) > 5, @GET(.color1), @GET(.color2) )  
)
```

## CASE

@CASE( [condition1]:expr, [if\_true]:expr, [condition2]:expr, [if\_true]:expr, ... )

v2.6.0+

The case statement is an extension of the IF method, but it allows any number of arguments. The odd arguments are conditional tests which must evaluate to True or False. The case method will go through each of the odd arguments until one of the evaluates to True at which point it will evaluate the corresponding even argument and return that value.

```
@CASE(  
  @GET(.age) < 10, 'blue',  
  @GET(.age) < 20, 'green',  
  @GET(.age) < 30, 'yellow',  
  @GET(.age) >= 30, 'red'  
)
```

## FOREACH

v2.6.0+

The following expression gets all the first name from the login table as a list. and then loop through and add <li> </li> around each item. This is more suited in situation where you don't much control over the data returned like in a CustomLayoutWdg:

```
@FOREACH( @GET(sthpw/login.first_name), '<li>%s</li>' )
```

## JOIN

@JOIN( [expr]:expression, [delimiter]:literal

v2.6.0+

The join method take the result of an expression and joins all the elements together using a delimiter

## UPDATE

@UPDATE( [expr1]:expression, [column]:string, [value]:expression )

v2.6.0+

The UPDATE method provides the ability for an expression to update a value in the database

The following example updates all of the modelling task to approved

```
@UPDATE( @SUBJECT(sthpw/task['context','model']), 'status', 'Approved' )
```

You can display a model task status column in the Asset page and any other asset related pages and have them all pointing back to the task search type during an update. It would eliminate any redundant data. The following xml definition can be used to set this up in the asset page for instance:

```
<element edit='true' name='asset_task_status' title='Task Status'>
  <display widget='expression'>
    <expression>@GET(sthpw/task['context','model'].status)</expression>
  </display>
  <action class='DatabaseAction'>
    <expression>@UPDATE(@SUBJECT(sthpw/task['context','model']), 'status', $VALUE) </
expression>
  </action>
</element>
```

The edit view for the Widget Config of prod/asset needs to contain this snippet to display the selection list of different statuses

```
<element name='asset_task_status'>
  <display class='tactic.ui.widget.TaskStatusSelectWdg' />
</element>
```

## EVAL

@EVAL( [expr1]:expression )

@( [expr1]:expression )

v2.6.0+

## Expression Variable Reference

There are a number of predefined variables in the expression language. The following list all of the available variables:

- LOGIN - the login of the current user
- PROJECT - code of the current project

(v2.6.0+)

- TODAY - the current day at midnight (12:00am)
- NOW - the current day and time
- NEXT\_DAY - the next day from today (tomorrow)
- NEXT\_MONDAY - the next Monday from today
- NEXT\_TUESDAY - the next Tuesday from today
- NEXT\_WEDNESDAY - the next Wednesday from today
- NEXT\_THURSDAY - the next Thursday from today
- NEXT\_FRIDAY - the next Friday from today
- NEXT\_SATURDAY - the next Saturday from today
- NEXT\_SUNDAY - the next Sunday from today
- PREV\_DAY - the previous day from today (yesterday)
- PREV\_MONDAY - the previous Monday from today
- PREV\_TUESDAY - the previous Tuesday from today
- PREV\_WEDNESDAY - the previous Wednesday from today
- PREV\_THURSDAY - the previous Thursday from today
- PREV\_FRIDAY - the previous Friday from today
- PREV\_SATURDAY - the previous Saturday from today
- PREV\_SUNDAY - the previous Sunday from today
- THIS\_YEAR - the first day of this year at midnight (12:00am)
- NEXT\_YEAR - the first day of next year at midnight (12:00am)
- PREV\_YEAR - the first day of the previous year at midnight (12:00am)
- THIS\_MONTH - the first day of this month at midnight (12:00am)
- NEXT\_MONTH - the first day of next month at midnight (12:00am)
- PREV\_MONTH - the first day of the previous month at midnight (12:00am)
- THIS\_HOUR - this hour at 0 minutes

- NEXT\_HOUR - the next hour at 0 minutes
- PREV\_HOUR - the previous hour at 0 minutes
- NEXT\_HOUR - the next hour at 0 seconds
- PREV\_HOUR - the previous hour at 0 seconds

These variables can be used for to refer to state information in searches. This expression will retrieve all the login information for the current user.

```
@GET(sthpw/login['login',$LOGIN])
```

They can also be used to find items between certain dates. This expression will retrieve all snapshots for this week starting at Sunday.

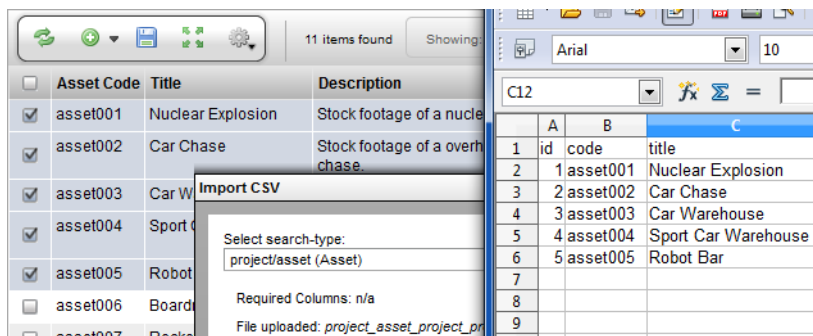
```
@GET(sthpw/snapshot['timestamp','>',$LAST_SUNDAY]['timestamp','<',$NEXT_SUNDAY])
```

The following variables are only used in Naming. Refer to the file naming section for details.

- EXT - file extension
- BASEFILE - the filename portion of the file without the extension

# Import and Export

## Importing CSV Data



Data can be imported into TACTIC and associated with the appropriate columns. Since TACTIC is based on a table system, data can be exported as a CSV file for easy manipulation in a spreadsheet program.

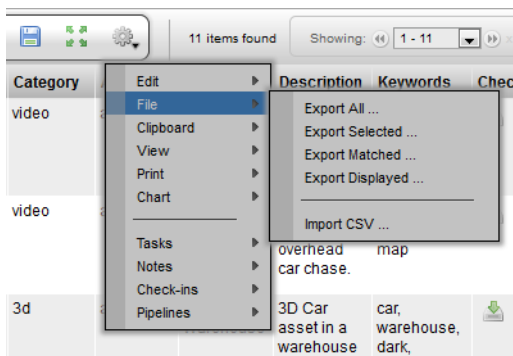


### Note

For more information on exporting existing CSV data, go to the section, "Exporting CSV Data."

#### To import a CSV file:

1. Go to **Gear Menu -> File -> Import CSV ...**



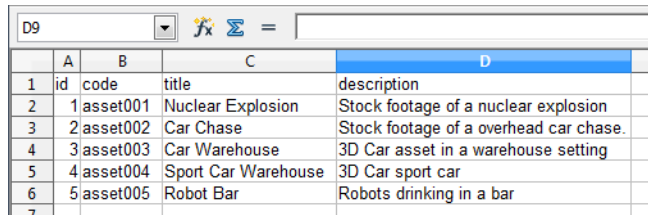
2. Select an sType. Once an sType is selected, the required columns will be displayed to the right of the drop down. Although these aren't the only columns that can be imported for this sType, they represent the minimum requirements for a successful import.



3. Browse and upload the CSV file. The CSV file is the raw format required by TACTIC. Below is an example of the contents of a CSV file:

```
"id","code","title","description"
1,"asset001","Nuclear Explosion","Stock footage of a nuclear explosion"
2,"asset002","Car Chase","Stock footage of a overhead car chase."
3,"asset003","Car Warehouse","3D Car asset in a warehouse setting"
4,"asset004","Sport Car Warehouse","3D Car sport car"
5,"asset005","Robot Bar","Robots drinking in a bar"
```

Below is an example of a CSV file being manipulated in a spreadsheet application.



	A	B	C	D
1	id	code	title	description
2	1	asset001	Nuclear Explosion	Stock footage of a nuclear explosion
3	2	asset002	Car Chase	Stock footage of a overhead car chase.
4	3	asset003	Car Warehouse	3D Car asset in a warehouse setting
5	4	asset004	Sport Car Warehouse	3D Car sport car
6	5	asset005	Robot Bar	Robots drinking in a bar
7				

- Click the Upload button. The TACTIC system will introspect the CSV file and fill in the rest of the import interface.



Import CSV

Select search-type:

project/asset (Asset)

Required Columns: n/a

File uploaded: project\_asset\_project\_project\_asset\_tracking3.csv

Change

The following is taken from first line in the uploaded csv file. Select the appropriate column to match.

Parsing Options

Use Title Row: ☒ ?

Sample Data Row: 1 ?

Encoder: ASCII (default)

Identifying Column: - Select - ?

<input checked="" type="checkbox"/>	CSV Column Value	TACTIC Column	Create New Column
<input type="checkbox"/>	1	id	
<input checked="" type="checkbox"/>	asset001	code	
<input checked="" type="checkbox"/>	Nuclear Explosion	title	
<input checked="" type="checkbox"/>	Stock footage of a nuclear explosion	description	

Preview Data

The following table will be imported into Asset (Showing Max: 100)

Refresh Import

id	code	title	description
1	asset001	Nuclear Explosion	Stock footage of a nuclear explosion
2	asset002	Car Chase	Stock footage of a overhead car chase.
3	asset003	Car Warehouse	3D Car asset in a warehouse setting
4	asset004	Sport Car Warehouse	3D Car sport car
5	asset005	Robot Bar	Robots drinking in a bar

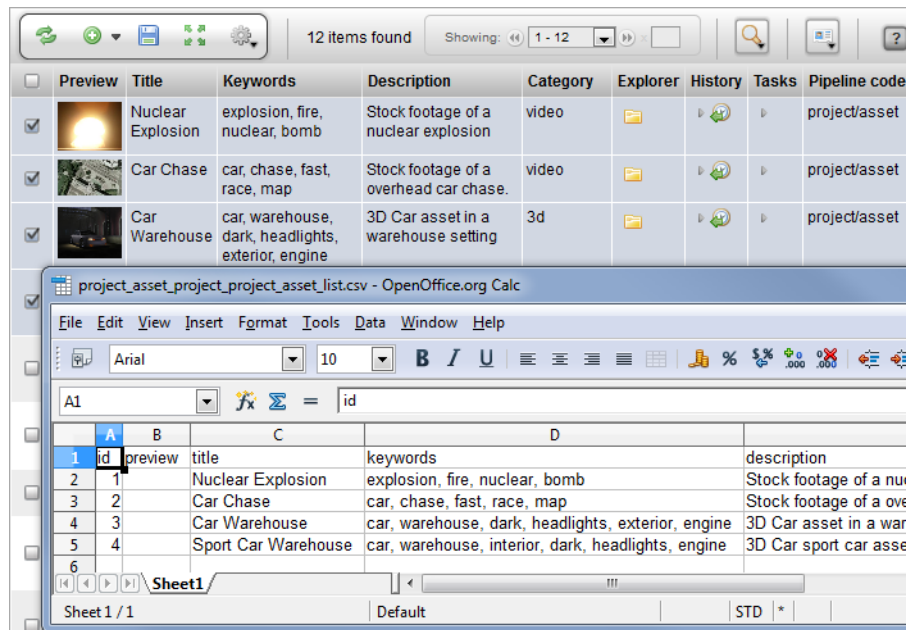
## 5. Parsing Options

<b>Use Title Row</b>	When checked, this import will read the first row in the file to determine which item columns are being imported.
<b>Sample Data Row</b>	Specify a row in the file to use as a sample for the data import.
<b>Encoder</b>	Select the encoding format for the data: ASCII (default), UTF-8, Excel ISO 8859-1
<b>CSV Column Value</b>	Displays the data from the sample data row you specified.
<b>Identifying Column</b>	Select the column which unique identifies the corresponding row in the table to update the data with.

## 6. Import

The bottom of the page shows a preview of the import results. Hitting the **Import** button will start the CSV import.

## Exporting CSV Data

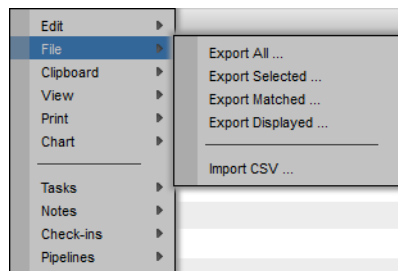


## Description

TACTIC uses a database and thus naturally exports to table-based outputs. TACTIC includes a simple user interface to export any table-based view out to a CSV file (Comma Separated Values file). Data can also be edited and imported to update or create new items in TACTIC.

## Exporting Data

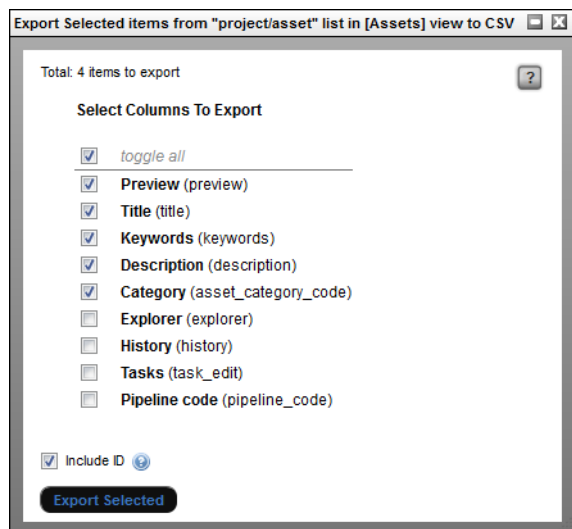
To export data from a TACTIC view to a CSV file, first check mark the rows to export. Then, go to the gear menu and under File select one of the following options:



Other export options include

- **Export All** - All items in the database
- **Export Selected** - Only the selected items in the view.
- **Export Matched** - All items found by the search
- **Export Displayed** - All items in the current view.

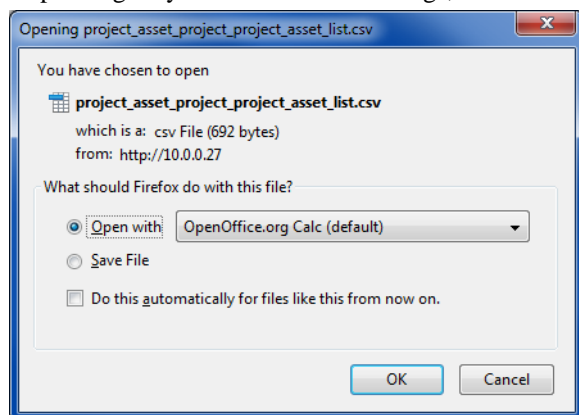
Next, select the columns to export from the selection list and click the **Export Selected** button.



## Note

Only columns saved to the current view will be available for export. It may be ideal to save a view particularly for exporting which includes all desired columns.

Depending on your web browser settings, the web browser will prompt to open or save the file.



Below is an example of the contents of the resulting CSV file:

```
"id","preview","title","keywords","description","asset_category_code"
"1","","Nuclear Explosion","explosion, fire, nuclear, bomb","Stock footage of a nuclear explosion","video"
"2","","Car Chase","car, chase, fast, race, map","Stock footage of a overhead car chase.", "video"
"3","","Car Warehouse","car, warehouse, dark, headlights, exterior, engine","3D Car asset in a warehouse setting","3d"
"4","","Sport Car Warehouse","car, warehouse, interior, dark, headlights, engine","3D Car sport car", "3d"
```

When the CSV file is open in a CSV-compatible spreadsheet application, you will be able to easily manipulate and add data:

## TACTIC Setup

---

	A	B	C	D	E	F
1	id	preview	title	keywords	description	asset_category_code
2	1		Nuclear Explosion	explosion, fire, nuclear, bomb	Stock footage of a nuclear explosion	video
3	2		Car Chase	car, chase, fast, race, map	Stock footage of a overhead car chase.	video
4	3		Car Warehouse	car, warehouse, dark, headlights, exterior, engine	3D Car asset in a warehouse setting	3d
5	4		Sport Car Warehouse	car, warehouse, interior, dark, headlights, engine	3D Car sport car asset in a internal warehouse setting	3d

# Advanced Configuration

## Modify Project Settings

Use the Project Settings tab to control the various options that exist within TACTIC. Most project settings are defined to work with the widgets that use them and when defined, the "Type" property specifies how the "Value" property is delivered to the widget. The different types of settings are outlined below.

- String - A single string argument, this may be a true/false to define how a widget is displayed (i.e. hide a specific aspect)
- Sequence - A sequence of items to choose from for entry (i.e. review|revise|complete)
- Map - A map is a sequence in which each item has a label and name assignment. This accommodates a separation between what is shown in a drop-down [name] vs what is entered into the database [label] (i.e. rvw:Review|rev:Revise|com:Complete)

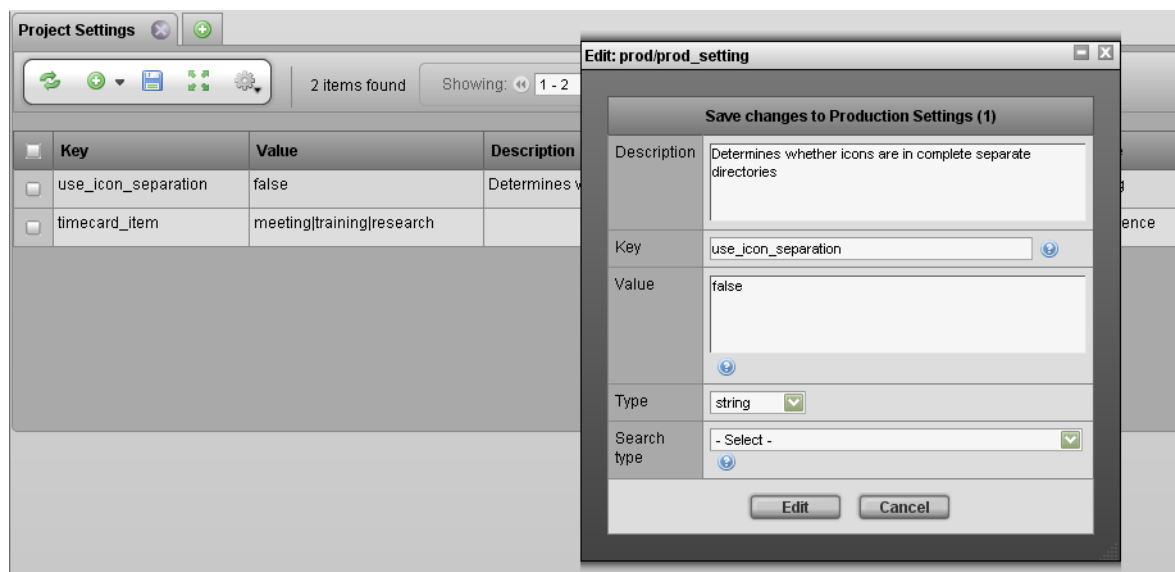


### Note

The overall items in the sequence or map are separated with a pipe '|' character and the value:label are separated with a Colon ':' character

Most settings are types of "sequences" that appear in TACTIC as a drop-down. For example, the notes\_dailies\_context setting defines the different kinds of context you can use in entering notes for dailies.

To insert a project setting, click the insert button in the view.



The properties for the project setting search type are listed below:

<b>Description</b>	A description of the purpose of the project settings
<b>Key</b>	This property serves as the 'code' identifier of the setting
<b>Value</b>	The Values for the setting.
<b>Type</b>	The 'type' of data definition of the value data. This tells the widget begin delivered the value how the data should be displayed.

<b>Search Type</b>	A search type to associate the project setting to, this help further filter the settings.
--------------------	---

Any widgets that make use of a new project setting not yet defined in TACTIC will prompt the user to insert data for a new project setting.

### Commonly Project Setting Examples

This table lists the some commonly used project settings in TACTIC.

key	Description	Default Value	Type
<b>flash_output_format</b>	Output format for a Flash project, swf swf OR mov		string
<b>fps</b>	Frames per second	24	string
<b>handle_texture_dependency</b>	Handle texture dependencies when true performing a checkin in a 3D application. Accepted values are 'true', 'false', 'optional'.		string
<b>notes_dailies_context</b>	Notes context used in the Dailies tab anim effects model	sequence	
<b>shot_hierarchy</b>	Shot hierarchy structure. Accepted sequence values are 'episode_sequence' or 'sequence'.		string
<b>bin_label</b>	Label for a Bin	n/a	string
<b>bin_type</b>	Type of Bin	n/a	string
<b>web_file_size</b>	dimension of the web type file size, 640x480 e.g. 640x480		string
<b>thumbnail_protocol</b>	The protocol through which the link http of a thumbnail is opened. Accepted values are 'file', 'http'.		string
<b>versionless_mode</b>	The global setting for copy or copy symlink for versionless check-ins		string

## Advanced Schema Configuration

When creating a search type, the "Search Type" property defines the project schema (project\_namespace) and name for the search type. For example, if your current project is called **media** then adding a new search type named "artwork" into the interface will automatically generate "media/artwork" and it will be added to the media project's schema

### Schema XML Structure

```
<?xml version='1.0' encoding='UTF-8'?>
<schema>
  <search_type name='media/training_videos' />
  <search_type name='media/script' />
  <connect to='media/script' type='hierarchy' from='media/training_videos' />
  <search_type name='media/fonts' />
  <search_type name='media/artwork' />
</schema>
```

To add a new search for a different schema, all that is required is an explicit definition of the full Search Type "<project\_namespace></name>".

For example to add an "artwork" search type in the "media" namespace, you would define the search type as "media/artwork". The specific schema information will be added to your current project only.



# Advanced Access Rule Configuration

## XML Access Rules

Security access rules are XML documents attached to user groups. Each user's security clearance is based in a union of all of the rules coming from the groups they are planned to. Admin users have no rules because, by default, everything is allowed in TACTIC.

In the Groups table, the Access Rules column shows the XML rules which are automatically generated by either the TACTIC toolset like the Permission Manager or manual editing for very specific and custom control. In general, the access level is ranked like this from bottom up: deny < view < edit < allow.

The following sample shows access rules applied to a 'Client' group:

In this sample, any users in the Client group can see only a project named "game" and cannot access the side\_bar items (which have been denied).

```
<?xml version='1.0' encoding='UTF-8'?>
<rules>
  <rule key='admin' access='deny' group='side_bar' />
  <rule key='site_admin' access='deny' group='side_bar' />
  <rule key='Level_Manage' access='deny' group='side_bar' />
  <rule key='levels_folder' access='deny' group='side_bar' />
  <rule key='characters_folder' access='deny' group='side_bar' />
  <rule key='myTactic_folder' access='deny' group='side_bar' />
  <rule group='project' access='deny' />
  <rule group='project' key='game' access='allow' />
</rules>
```

## XML Examples

The following are examples of different access rules which can be used to customize group access rules. Make sure the <rule/> tag is a child of the <rules/> tag.

### Project level Examples

1. This rule denies access to all projects except for the "sample3d" project. In the following example, the "default" project is a home page the user needs to use to select projects. Because it is part of the group, you must explicitly allow viewing access to this default project when you deny access to all projects. It is also needed for XML-RPC communication to the client computer.

```
<rules>
  <rule group='project' default='deny' />
  <rule group='project' key='sample3d' access='allow' />
  <rule group='project' key='default' access='view' />
</rules>
```

2. All projects can be viewed by default.

```
<rule group='project' default='view' />
```

### Search Type level Examples

1. The layer search type is not viewable in all projects

```
<rule group='subject' search_type='prod/layer' project='*' access='deny' />
or
<rule group='subject' search_type='prod/layer' access='deny' />
```

2. The task search type is not viewable in project 'sample3d'. Note that this is not the same as tasks assigned in project 'sample3d' is not viewable. It merely restricts the user's ability to view tasks when he is in a particular project.

```
<rule group='subject' search_type='sthpw/task' project='sample3d' access='deny' />
```

3. The note search type is not viewable in project 'sample3d'.

```
<rule group='subject' search_type='sthpw/note' project='sample3d' access='deny' />
```

4. The note search type is not editable in project 'sample3d'. This currently only applies to the main TableLayoutWdg used in most places. NoteSheetWdg and DiscussionWdg which also handle note entry are not bound by this rule.

```
<rule group='subject' search_type='sthpw/note' project='sample3d' access='view' />
```

5. The shot search type is editable in project 'sample3d'

```
<rule group='subject' search_type='prod/shot' project='sample3d' access='edit' />
```

6. The 3d Asset search type from project 'sample3d' is not viewable. This is also applicable when you are in a different project looking at a task in 'sample3d' and the parent of which happens to be a 3d Asset in project 'sample3d'.

```
<rule group='subject' search_type='prod/asset' project='sample3d' access='deny' />
```

### Search Type Column level Examples

These examples affect the display of the columns in different views

1. The 3d Asset search type's code and description are not editable in all projects

```
<rule group='element' search_type='prod/asset' key='code' access='view' />  
<rule group='element' search_type='prod/asset' key='description' access='view' />
```

2. The Shot search type's status is not editable in the 'sample3d' project

```
<rule group='element' search_type='prod/shot' key='status' access='view' project='sample3d' />
```

3. The Shot search type's description is not visible in the 'sample3d' project

```
<rule group='element' search_type='prod/shot' key='description' access='deny'  
project='sample3d' />
```

### Database level Examples

While Search Type and Search Type Column level examples affect the display of the main TableLayoutWdg and EditWdg, the following database level examples are applied when attempts are made to edit or insert data into the database. It can block even server or client API script access to the databases.

1. This rule prevents the display and writing of the "is\_current" field for snapshots found in the Checkin History.

DEPRECATED and UNSUPPORTED format:

```
<rule group='subject|column' key='sthpw/snapshot|is_current' access='deny' />
```

New format:

```
<rule group='subject_column' search_type='sthpw/snapshot' column='is_current'
access='deny' />
```

2. The description column for Shot cannot be edited in any widget or any script. It is view only.

```
<rule group='subject_column' column='description' search_type='prod/shot' access='view' />
```

3. The status column for Task cannot be edited in any widget or any script. It is view only.

```
<rule group='subject_column' column='status' search_type='sthpw/task' access='view' />
```

4. The custom sType project/asset is view-only and not editable by a particular group.

```
<rule group='subject' search_type='project/asset' access='view' />
```

5. The custom sType project/asset is not viewable by a particular group. The search result will always come up empty.

```
<rule group='subject' search_type='project/asset' access='deny' />
```

## Search Filter Examples

To enforce what can be searched or filtered out in any situation like script query or UI view, search\_filter rules can be applied. The 'access' attribute is not required here.

1. This rule filters out tasks belonging to project "pacman" and "sample3d". Notice you don't need "access" here.

```
<rule group='search_filter' column='project_code' value='pacman' op='!=' search_type='sthpw/
task' />
<rule group='search_filter' column='project_code' value='sample3d' op='!='
search_type='sthpw/task' />
```

2. This rule retrieves task that is assigned to the current login user, applicable when navigating in project 'sample3d'. Notice the 'value' attribute can accept an expression. \$LOGIN and \$PROJECT are also supported.

```
<rule column='assigned' value='@GET(login.login)' search_type='sthpw/task' op='='
group='search_filter' project='sample3d' />
or
<rule column='assigned' value='$LOGIN' search_type='sthpw/task' op='=' group='search_filter'
project='sample3d' />
```

## Miscellaneous Examples

1. This rule blocks a user from seeing the options "Approved" and "Complete" in the task status drop-down

```
<rule access='deny' key='Complete' category='process_select' />  
<rule access='deny' key='Approved' category='process_select' />
```

## Remove Projects

There is no user interface in TACTIC to remove a project. This is because the complete removal of a project has some pretty significant consequences. When a project is created, a number of elements are created. These are listed below.



### Note

This task may need to be carried out by the Tactic System Admin as it involves manually accessing both the Tactic File system and the Tactic Database

In all of the following examples, <project\_code> represents the code of the project when the project was created.

- A database called <project\_code>
- An assets directory in <tactic\_asset\_dir>
- Entry in the project table

**A complete removal of a project should be handled with care. This is most often desirably when a project has been created in properly. It is one of the few operations that is not undo-able in TACTIC, so it is recommended to be careful when proceeding with the following steps. It is also recommended that a complete backup of TACTIC is performed before carrying out this process.**

### Remove the project directory

```
cd <project_install_dir>/sites
rm -rf <project_code>
```

### Remove the database

```
psql -U postgres sthpw
drop database "<project_code>";
```

### Remove the assets directory

```
cd <project_asset_dir>
rm -rf <project_code>
```

### Remove the entry in the projects table in the database

```
psql -U postgres sthpw
delete from project where code='<project_code>;'
```

### Remove connected entities in the sthpw database;

In this process, Tactic may not allow removing a particular project due to there being child tasks, notes, snapshots, files, wdg\_settings etc. If Tactic denies removal because, for example there is a connection in the file table, you will need to do the following

```
delete from file where project_code='<project_code>;'
delete from snapshot where project_code='<project_code>;'
delete from task where project_code='<project_code>;'
delete from note where project_code='<project_code>;'
delete from wdg_settings where project_code='<project_code>;'
delete from pref_setting where project_code='<project_code>;'
```

# Advanced Automation

## Advanced Notification Setup

The notification view is located in the TACTIC Sidebar under:

**Site Admin -> Notifications**

This can also be accessed through the **Workflow Editor**, by right clicking on a node and selecting from the context menu **Show Triggers/Notifications**.

This view provides all the functionality required to set up the various types of notifications used to establish better production communication and instant status updates.

Notifications					
<div> <div> </div> <div>6 items found</div> <div> Showing: 1 - 6 </div> <div> </div> </div>					
<input type="checkbox"/>	Email test	Event	Description	Subject	Message
<input type="checkbox"/>	Email Test	update sthpw task status	Notify when the status changes.	TACTIC: {@GET(parent.code)} [[{@GET(.context)}]] Updated to {@GET(.status)}	{@GET(parent.code)} has  Task:{@GET(.context)} Assigned: {@GET(.assigned)} Status: {@GET(.status)}
<input type="checkbox"/>	Email Test	update sthpw task status	Notify when the status is set to review.	TACTIC: {@GET(parent.code)} Ready for Review	{@GET(parent.code)} is Review
<input type="checkbox"/>	Email Test	update sthpw task assigned	Notify when a task is assigned to you.	TACTIC: New task assignment {@GET(parent.code)} [[{@GET(.context)}]]	{@GET(parent.code)} has to you.  Task:{@GET(.context)} Assigned: {@GET(.assigned)} Status: {@GET(.status)}

## Insert Notifications

To insert a new notification, select from the sidebar:

**Site Admin -> Notifications**

The following table explains the basic usage of each property.

<b>Event</b>	The TACTIC event to execute the notification for
<b>Process</b>	Events relating to note, task, snapshot can make use of process to differentiate
<b>Description</b>	The description of the purpose of the notification
<b>Subject</b>	The subject of the notification. This uses the TACTIC Expression Language
<b>Message</b>	The message body for the notification. This uses the TACTIC Expression Language
<b>Rules</b>	The XML access rules used to filter the notification further
<b>Email Handler Class</b>	The email handler class override for the notification. This allows for overriding of the notification with a python email handler class. This only needs to be used in specific situations.
<b>Project Code</b>	The project code for the notification. This allows for filtering of notifications for a specific project
<b>Type</b>	The type of notification being sent. By default for notifications, this must be set to "email"
<b>Mail_to</b>	An expression of the users to mail to in the email (supports multiple lines of expressions and / or email addresses and names of groups of users made in TACTIC)
<b>Mail_cc</b>	An expression of the users to mail to in the email. It will appear in the cc category. (supports multiple lines of expressions and / or email addresses and groups)
<b>Mail_bcc</b>	An expression of the users to mail to in the email. It will appear in the bcc category. (supports multiple lines of expressions and / or email addresses and groups)

Below are examples of what can be used in *mail\_to*, *mail\_cc*, or *mail\_bcc*:

For example, email the user 'admin' only:

```
@SUBJECT(sthpw/login['login', 'admin'])
```

For example, send to the user related to this sObject. If this is an event for task update, the email will be addressed to the assigned user:

```
@SUBJECT(subject.sthpw/login)
```

For example, send to the user related to this sObject as well as the user 'john'. If the event is insert|sthpw/note, it will be the person who enters the note. If the event is 'update|sthpw/task', the person is the assignee of the task:

```
@UNION(@SUBJECT(sthpw/login['login', 'john']), @SUBJECT(subject.sthpw/login))
```

To make these mail\_to expressions more readable, put more than 1 expression or email addresses on multiple lines. There is no need for @UNION. @GET can even be used to just get to the list of login names.

For example, to send to everyone in the supervisor group, the assignee of a task, to all the users in the **mangers** group and the email address support@southpawtech.com, enter 3 lines under mail\_to:

```
@SUBJECT(subject.sthpw/login)
@GET(sthpw/login_in_group['login_group', 'supervisor'].login)
mangers
support@southpawtech.com
```

## More on Expressions in Notifications

The word "subject" often appears in the *Mail to:* column but not in Message or Subject. This is because the implementation allows sending notifications to users related to the current sObject or just about anyone not necessarily related to the current sObject. As illustrated above, @SUBJECT(subject.sthpw/login) is the task assignee but the users under the group supervisor is not related to this task and so the keyword "subject" is not used. In the Message area, to refer to the current sObject status (task status if the event is update|sthpw/task), just use an @GET(.status), as the subject is always assumed to be in this context.

### Task related notification

To establish the relationship between the login search type and task search type, the following built-in schema line is used. It is not necessary to add it to the schema. It can be used as an example to create a custom search\_type.

```
<connect to='sthpw/task' relationship='code' from_col='login' from='sthpw/login'
to_col='assigned' />
```

### Note related notification

To establish the relationship between the login search type and note search type, the following built-in schema line is used. It is not necessary to add it to the schema. It can be used as an example to create a custom search\_type and to edit the schema.

```
<connect to='sthpw/note' relationship='code' from_col='login' from='sthpw/login' to_col='login' />
```

Sending a notification to the person who just entered the note is not often used. Instead, an email handler can be used in this situation to send to the supervisor and assignee of the task under the same context. A built-in email handler is called "pyasm.command.NoteEmailHandler". Instead of entering it into an expression for *mail\_to*, enter it into the *email\_handler\_class* field.

## Email Test

<input type="checkbox"/> Email test	Event	Description	Subject	Message
<input type="checkbox"/> <b>Email Test</b>	update sthpw/task status	Notify when the status is set to review.	TACTIC: { @GET(parent.code) } Ready for Review	{ @GET(parent.code) } is ready for Client Review

Once the fields *event*, *mail\_to*, and *message* are properly filled in, to test the email, click on the **Email Test** button. It catches syntax errors or typos in expression in these fields as well as reporting any email server error if the service section of the TACTIC config file has not been properly filled in. Settings like firewall and TLS settings may also block an email from being sent out.

### Example 1:

In this example, the notification will be sent out each time a ticket is updated. It will also only send to users in the 'admin' and 'moderator' groups.

<b>Event</b>	update support/ticket
<b>Description</b>	Sent when tickets are updated
<b>Subject</b>	TACTIC Ticket { @GET(.id) } Has been updated.
<b>Message</b>	{ @GET(.company_code) } TACTIC Ticket { @GET(.id) } has been updated.  Subject: { @GET(.subject) }



	<pre>Summary: {@GET(.summary)}  Status:  {@GET(.status)}  From status: {@GET(sthpw/status_log['@LIMIT','1'].from_status)}  Please do not reply to this e-mail.  To reply or to make further comments, please login to the Ticketing system @ http://tickets.southpawtech.com/tactic</pre>
<b>mail_to</b>	admin  moderators
<b>Rules</b>	
<b>Email Handler Class</b>	
<b>Project Code</b>	support
<b>Type</b>	email

## Example 2:

In this example, the notification will be sent out each time a shot-based note is updated. It will send to manager's group and everyone assigned to the tasks of the shot. Since project\_code is left empty, this works across all the projects in the system.

<b>Event</b>	insert sthpw/note
<b>Description</b>	Sent when shot-based notes are added
<b>Subject</b>	{\${PROJECT}} {@GET(parent.sequence_code)} - {@GET(parent.code)} {@GET(.process)}
<b>Message</b>	<pre>Project: {\${PROJECT}} Code: {@GET(parent.sequence_code)}, {@GET(parent.code)} Process: {@GET(.process)} Note: {@GET(.note)}</pre> <p>To reply or make further comments: please go to <a href="http://&lt;some IP&gt;/tactic/{\${PROJECT}}/#/link/my_notes">http://&lt;some IP&gt;/tactic/{\${PROJECT}}/#/link/my_notes</a></p>
<b>Rules</b>	<pre>&lt;rules&gt;   &lt;rule&gt;'prod/shot' in @GET(.search_type)&lt;/rule&gt; &lt;/rules&gt;</pre>
<b>Email Handler Class</b>	
<b>Project Code</b>	
<b>Type</b>	email
<b>mail_to</b>	<pre>@GET(subject.parent.sthpw/task.assigned) @GET(sthpw/login_in_group['login_group','manager'].login)</pre>

The following sections explain each configuration aspect of Notifications in greater detail

## Notification Expressions

TACTIC uses the TACTIC Expression Language to build dynamic Notification Subject and Message contents. This allows for each notification to be sent based on properties from the Search Objects it is being sent for.

In the simple example Subject below, the "id" property is used from the "ticket" search object.

```
TACTIC Ticket {$GET(.id)} has been updated
```

The expected results of this would be similar to the following:

"TACTIC Ticket 14 has been updated"

### Example 1

In essence, anything between the curly brackets "{}" is evaluated as an expression by TACTIC.



### Note

For more information regarding TACTIC Expression Language please refer to the **TACTIC Expression Language** docs

## Filtering Notifications

TACTIC's notification architecture is a rules-based system built using the trigger architecture. Every time a command is executed, TACTIC looks through the list of defined triggers (including notifications) for a match. Under the **Triggers** view will be an entry for the EmailTrigger class that is registered under the "email" event. It is possible to create custom Email Trigger handlers in that view.

There are 3 main criteria used to filter out notifications:

- **group:** Filters out notifications to be sent only to users in the included groups.
- **project:** Filters out notifications so that only a certain project can fire the email trigger.
- **rules:** Rules are an XML snippet which can finely control the conditions when an email trigger may be fired.

## Groups

By planning groups to send notifications to, it allows for simple connections for deciphering which groups of users will receive notifications when the conditions of a particular notification rule are met. Once a notification has been created, it can be associated with any number of groups of users. All users in this group will then be sent a notification when the rule is triggered.

The Groups view can be found under:

**Admin->Site Admin ->Groups**

To specify a group to send a notification to, specify the group in the **mail\_to** column.

### Project

By setting the project in the project column of a Notification, TACTIC will only use the notification trigger for the chosen project.

## Access Rules

When a notification rule has passed all of the criteria, a message is constructed. Most email events occur after a command has been completed. The email handler then takes the information from the command and creates a default message to be sent to the appropriate people.

All rules are contained in groups. For notifications, there are a few predefined groups:

### Example 1

This rule group only allows tasks for prod/asset for the project sample3d to send out notification. Otherwise, it would send out notifications for tasks of all search types

```
<rules>
  <rule>@GET(.search_type)=='prod/asset?project=sample3d'</rule>
</rules>
```

### Example 2

This rule group makes use of a key/value pair of attributes: that is, when the attribute with the value of "key" is equal to "value", the rule is passed. In the example below, all SObjects containing the attribute "context" with the value "client" are triggered.

```
(deprecated)

<rules>
  <rule group="SObject" key="context" value="client"/>
</rules>

<rules>
  <rule>@GET(.context)=='client'</rule>
</rules>
```

### Example 3

For certain SObjects in TACTIC (like tasks), parent attributes can be used for constructing rules. The concept behind this is the same as group="sObject", but now we are referring to the parent of a task (for example, a 3D asset). This notification will only be sent if the task's parent, a 3D asset, is categorized under the "prp" asset library.

```
<!-- DEPRICATED -->
<rules>
  <rule group='parent' key='asset_library' value='prp' />
</rules>

<rules>
  <rule>@GET(prod/asset.asset_library)=='prp'</rule>
</rules>
```

### Example 4

For notes in TACTIC, we may have 2 processes for notes (e.g. anim, anim\_2) We can check if the process partially contains the word **anim** by the following:

```
<rules>
  <rule>'anim' in @GET(.process)<rule>
</rules>
```

Note: list comparisons like @GET(.process) in ['anim','anim\_2'] are not supported

### Example 5

For a check-in notification in TACTIC, we can choose to send only if the is\_current attribute is True for the event insert|sthpw/snapshot by the following:

```
<rules>
  <rule>@GET(.is_current)==True<rule>
</rules>
```

## Email Handler Class

Each time a notification is executed, TACTIC uses either the default email handler or it uses an email handler override defined by the Email Handler Class property for the notification.

The Email Handler Class digs deeply into the structure of the notifications using Python and the TACTIC client API. It is only needed for very specific rules which determine when a notification is sent.

An example override is shown below:

**Email Handler Cls:** sites.support.email.TicketEmailHandler

```
__all__ = ['TicketEmailHandler']

from pyasm.common import Environment, Xml, Date
from pyasm.security import Login
from pyasm.search import Search
from pyasm.biz import GroupNotification, Pipeline, Task, Snapshot, File, Note

class TicketEmailHandler(object):
    '''Email sent when a ticket is updated'''

    def __init__(my, notification, SObject, parent, command):
        my.notification = notification
        my.sobject = SObject
        my.command = command
        my.parent = parent

    def check_rule(my):
        '''determine whether an email should be sent'''
        return True

    def get_to(my):

        ticket = my.sobject
        user = ticket.get_value("login")
        login = Login.get_by_login(user)
        recipients = []
        recipients.append(login)

        return recipients
```

```
def get_cc(my):

    admin = Login.get_by_login("admin")
    recipients = []
    recipients.append(admin)

    return recipients

def get_subject(my):

    ticket = my.sobject
    title = "Ticket Number: "
    id = ticket.get_value("id")

    return "%s%s" % (title, id)

def get_message(my):

    ticket = my.sobject
    id = ticket.get_value("id")
    subject = ticket.get_value("subject")
    summary = ticket.get_value("summary")
    status = ticket.get_value("status")

    msg = []
    msg.append("Ticket: %s" % id)
    msg.append("")
    msg.append("Status: %s" % status)
    msg.append("")
    msg.append("subject: %s" % subject)
    msg.append("Summary: %s" % summary)
    msg.append("")

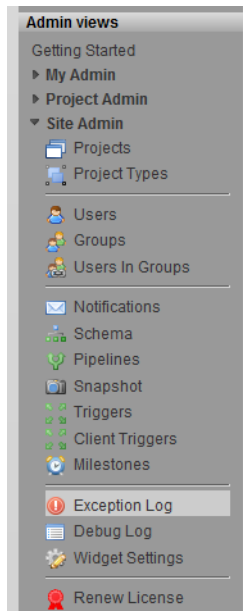
    return "\n".join(msg)
```

# Debugging TACTIC

## Exception Log

The exception log SObjects provides a very convenient way for TACTIC to store, view and diagnose problems or errors in TACTIC as they come up. This is a necessary requirement in order to properly diagnose a bug or error that will arise.

A default view of the exception log is available in the "Admin" section of the Schema Sidebar



Because TACTIC records every stack trace in the system, it is possible to simply find the last stack trace generated by a particular user. This eliminates the need to constantly reply to an error report asking for the stack trace of the error (which by then is often lost because the user has decided to work on another task). With the exception log, it becomes trivial to look up the error witnessed by the user and start diagnosing the problem.

# TACTIC Setup

Exception Log

284 items found Showing 1-20

	Id	Login	Timestamp	Class	Message	Stack trace
	1	admin	Jul 04, 2010	SearchException	Search type [supportTicket] not registered	<p>Error parsing xml [in XmlWrapper, line 1, column 2: syntax error]:</p> <pre>File "/home/apache/tactic/src/tactic/ui/app/page_nav_container_wdg.py", line 347, in get_display     main_body_content = Common.create_from_class_path(main_body_handler, [], main_body_options) File "/home/apache/tactic/src/pyasm/common/common.py", line 59, in create_from_class_path     object = marshaller.get_object() File "/home/apache/tactic/src/pyasm/common/common.py", line 524, in get_object     object = eval("%s(**my.kwargs)" % (unique_class_name)) File "&lt;string&gt;", line 1, in &lt;module&gt; File "/home/apache/tactic/src/tactic/ui/panel/layout_wdg.py", line 139, in __init__     super(TableLayoutWdg,my).__init__(search_type=my.search_type, config_base=my.view, config=config) File "/home/apache/tactic/src/pyasm/widget/layout_wdg.py", line 64, in __init__     my.search_type_obj = SearchType.get(search_type) File "/home/apache/tactic/src/pyasm/search/search.py", line 3173, in get_search_type     search_type = cls._get_data(base) File "/home/apache/tactic/src/pyasm/search/search.py", line 3330, in _get_data     raise SearchException("Search type [%s] not registered" % search_type )</pre>
	2	admin	Oct 25, 2010	AttributeError	'unicode' object has no attribute 'strftime'	<p>Error parsing xml [in XmlWrapper, line 1, column 2: syntax error]:</p> <pre>File "/spt/tactic/tactic/src/pyasm/prod/service/api_xmlrpc.py", line 3235, in get_widget     html = widget.get_buffer_display() File "/spt/tactic/tactic/src/pyasm/web/widget.py", line 414, in get_buffer_display     return my.get_buffer(cls).getvalue() File "/spt/tactic/tactic/src/pyasm/web/widget.py", line 406, in get_buffer     my.explicit_display(cls) File "/spt/tactic/tactic/src/pyasm/web/widget.py", line 364, in explicit_display     child = my.get_display() File "/spt/tactic/tactic/src/tactic/ui/panel/layout_wdg.py", line 1519, in __init__</pre>

## Submitting a Support Ticket

To submit an official support ticket to the Southpaw Support Team you will need a **TACTIC Ticketing Account**. Ticketing accounts are typically setup for users by Southpaw as part of official TACTIC support. Contact your TACTIC representative for more information.

You may also visit the Southpaw Technology Support site at:

**[support.southpawtech.com](https://support.southpawtech.com)**



### Note

If you do not already have an account for the support site, you may sign up from the login page. Accounts on the support site need to be verified by the support team. This typically takes about 24 hours.



# TACTIC Widgets

## TACTIC Widgets

### What is a Widget?

A TACTIC Widget is a mini web program which serves a specific purpose and can be reused. This is what the TACTIC interface is composed of. Each interface view is presented to a user or a group of users in a combination of widgets which can accomplish many things:

- Displaying data
- Modifying data
- Displaying images
- Executing processes
- Loading files
- Saving files
- ...and much more

Each of the widgets is a "snippet" of web code (HTML) which TACTIC builds dynamically on the server and delivers to the user as a web page; but the TACTIC interface is much more than a web page. TACTIC takes full advantage of the newest web technology to deliver a flexible and robust web application.

This widget architecture allows TACTIC to provide a toolbox of useful widgets which can be combined to build an interface which can adapt TACTIC to an work flow. Furthermore, due to the openness of TACTIC, it is easy for developers to create new widgets and share them.

### Widget Documentation

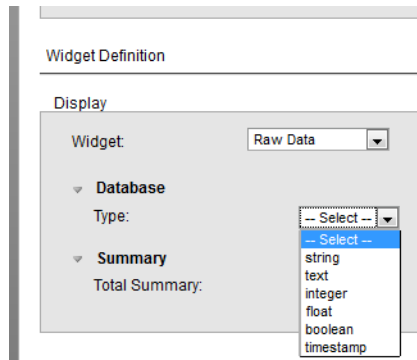
Within this documentation, the same philosophy will be followed. There is a section which explains the usage of each widget but the main core of the documentation explains how to use them in combination or with existing interfaces to work with TACTIC.

A few examples of the common TACTIC widgets are:

- ThumbWdg - A thumbnail viewing widget used in a table, usually with a title **preview**.
- TableLayoutWdg - Majority of the TACTIC interface is delivered through a Table widget. It enables the assembly of different type to save as a new view.
- GanttWdg - A calendar widget which allows viewing and manipulation of dates.

# View Widgets

## Simple Table Element



### Description

This widget displays the value in the database "as is", in its raw unformatted form. This is the default display widget.



#### Note

The Simple Table Element widget is the same as the Raw Data widget.

### Info

<b>Name</b>	Simple Table Element
<b>Class</b>	raw_data
<b>TACTIC Version Support</b>	3.0.0 +
<b>Required database columns</b>	none

### Implementation

Use this widget to display the value in the database "as is", without any pre-formatting. This widget is the default display widget.

### Options

<b>type</b>	The database type: string, text, int, float, boolean, timestamp
-------------	---

### Example

For example, to display the keywords field, as is, from the Edit Column Definition->View Mode: Select **Widget -> Raw Data -> text**.

### Advanced

Below is the XML for the above example.

```
<element name="keywords" title="test" edit="true" color="false">  
  <display widget="raw_data">  
    <type>text</type>  
  </display>  
</element>
```

## Formatted Widget

Cost budget	Cost total	Variance	Timestamp
\$0.00	\$436.78	(\$436.78)	Jul 29, 2011 - 01:49
\$0.00	\$0.00	\$0.00	Jul 29, 2011 - 00:21
\$14,000.00	\$2,170.00	\$11,830.00	Jul 28, 2011 - 06:46
\$7,000.00	\$4,969.00	\$2,031.00	Jul 28, 2011 - 06:28
\$5,000.00		\$5,000.00	Jul 26, 2011 - 20:01
\$2,000.00		\$2,000.00	Jul 26, 2011 - 20:01
\$8,000.00	\$14,750.00	(\$6,750.00)	Jul 26, 2011 - 20:01
\$7,200.00		\$7,200.00	Jul 26, 2011 - 20:01
\$500.00	\$700.56	(\$200.56)	Jul 26, 2011 - 20:01
\$10,000.00	\$5,600.00	\$4,400.00	Jul 26, 2011 - 20:01

## Description

The Formatted Widget displays a raw data value as a formatted string so the user can recognize and interpret the value more easily.

For example, the Format Widget will display a number in the format \$1,234.00 which the user quickly recognizes as a currency value in dollars and cents.

## Info

<b>Name</b>	Formatted Widget
<b>Class</b>	FormatElementWdg
<b>Category</b>	Simple Table Element Widget
<b>Supported Interfaces</b>	
<b>TACTIC Version Support</b>	3.6.0+
<b>Required database columns</b>	none

## Options

**Create New Column**

View Mode Edit Mode

**New Column**

Title:

Name:

Width:

**Widget Definition**

Display

Widget: Formatted

Database

Type: integer

Summary

Total Summary: -- Format --



integer  
float  
percent  
currency  
date  
time  
scientific  
boolean  
text  
timecode

Next > Create >>

<b>integer</b>	-1234
	-1,234
<b>float</b>	-1234.12
	-1,234.12
<b>percent</b>	-13%
	-12.95%
<b>currency</b>	-\$1,234
	-\$1,234.00
	-\$1,234.--
	-1,234.00
<b>date</b>	31/12/99
	December 31, 1999
	31/12/1999
	Dec 31, 99

	Dec 31, 1999
	31 Dec, 1999
	31 December 1999
	Fri, Dec 31,99
	Fri 31/Dec 99
	Fri, December 31, 1999
	Friday, December 31,1999 12-31
	99-12-31
	1999-12-31
	12/99
	31/Dec
	December
	52
<b>time</b>	13:37
	13:37:46
	01:37 PM
	01:37:46 PM
	31/12/99 13:37
	31/12/99 13:37:46
<b>scientific</b>	-1.23E+03
	-1.234E+03
<b>boolean</b>	true false
	True False
	Checkbox
<b>timecode</b>	MM:SS:FF
	MM:SS:FF
	HH:MM:SS:FF
	HH:MM:SS:FF

## Expression

Total Deposit	Total Cost	Cost balance due	Deposit Date	Payment Due Date
100	450	350	2010-05-05 	2010-05-11 

## Description

The ExpressionElementWdg allows you to use the TACTIC expression language to determine the value displayed in the table cell. The expression is calculated from a starting subject which represents the subject in the particular row in the table. The expression is evaluated for each subject on every row. When an expression is evaluated, the value is added to a dynamic attribute of the subject and can be used in future expressions in this widget. Please refer to the expression language reference for more information on the expression language.

## Info

<b>Name</b>	ExpressionElementWdg
<b>Class</b>	tactic.ui.table.ExpressionElementWdg
<b>Category</b>	Common Columns
<b>Supported Interfaces</b>	TableLayoutWdg
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	depends on expression

## Implementation

Display the total cost of an item by multiplying the total\_number column with the unit\_cost column. When an expression is evaluated by the ExpressionElementWdg, a new attribute with the name of the element is dynamically added to the subject (in this cost) which can be used in the "bottom" directive.

```
<element name='cost'>
  <display class='tactic.ui.table.ExpressionElementWdg'>
    <expression>@GET(.total_number) * @GET(.unit_cost)</expression>
    <bottom>@SUM(.cost)</bottom>
  </display>
</element>
```

## Options

<b>expression</b>	Expression to evaluate the widget
<b>alt_expression</b>	Alternate expression to evaluate the widget for alternate use, like edit value
<b>inline_styles</b>	Styles to add to the DIV generated that contains the result of the expression
<b>return</b>	single list - Determines what the expression return type should be
<b>bottom</b>	Expression to calculate the bottom row of the table
<b>mode</b>	value boolean check - Display mode for this widget
<b>enable_eval_listener</b>	Currently javascript expression evaluation is not fully baked, so only use the client side evaluation listener when needed and NOT by default

**icon\_expr**

Expression to evaluate which icon to use when mode = 'icon'

## Examples

Display the number of tasks for a given subject and then display the total number at the bottom.

```
<element name='num_tasks'>
  <display class='tactic.ui.table.ExpressionElementWdg'>
    <expression>@COUNT(sthpw/task)</expression>
    <bottom>@SUM(.num_tasks)</bottom>
  </display>
</element>
```

Mode "boolean" displays a green dot for every subject that has an expression that evaluates to True. In this case, a green dot is displayed on every row where the number of tasks is greater than zero.

```
<element name='has_tasks'>
  <display class='tactic.ui.table.ExpressionElementWdg'>
    <expression>@COUNT(sthpw/task) > 0</expression>
    <mode>boolean</mode>
  </display>
</element>
```

Another example of a mode which displays a checkbox instead of red/green dots. The checkbox appears for any result greater than zero

```
<element name='has_tasks'>
  <display class='tactic.ui.table.ExpressionElementWdg'>
    <expression>@COUNT(sthpw/task) > 0</expression>
    <mode>check</mode>
  </display>
</element>
```

The expression language has the ability to get values from other related tables. The following example illustrates an expression to find the description of the parent sequence of a shot.

```
<element name='sequence_description'>
  <display class='tactic.ui.table.ExpressionElementWdg'>
    <expression>@GET(prod/sequence.description)</expression>
  </display>
</element>
```

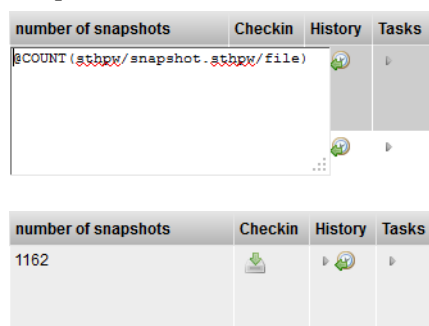
Ultimately, the ExpressionElementWdg can make use of any expression in the TACTIC Expression Language.

When using mode = 'icon', it is possible to set up an expression using icon\_expr to determine what that icon should be. A special variable \$VALUE is used to determine the value of the expressions



```
<element name="is_synced" title='Synced' edit='false'>
  <display class='tactic.ui.table.ExpressionElementWdg'>
    <expression>@GET(.is_synced) == True</expression>
    <mode>icon</mode>
    <icon_expr>@IF( '$VALUE' == True, 'CHECK', 'CROSS' )</icon_expr>
  </display>
</element>
```

## Expression Value Element



### Description

The Expression Value Element widget accepts a TACTIC Expression as the input and displays the evaluated expression as the output.

#### Info

<b>Name</b>	Expression Value Element Widget
<b>Class</b>	expression_value
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	Yes, a database column by the same name.

### Usage

For example, we can dynamically display the number of login names in the login table. This would be an example of an absolute expression because the expression does not take into input any data from the row the field is on. A relative expression has access to the row and table information that the row the expression is on.



#### Note

The difference between an **absolute expression** and a **relative expression**:

- an **absolute expression** does not take into input any data from the row or table that the field exists on
- a **relative expression** has access to the row and table information that the field exists on

### Implementation

Go into edit mode for the Expression Value Element widget. Input an absolute TACTIC expression as the value.

In display mode, this widget will display the result of the evaluation of the expression.

#### Options

There are no options available for this widget.

### Example 1

For example, enter the following absolute TACTIC Expression as the value for the Expression Value Element widget:

```
@COUNT(sthpw/snapshot.sthpw/file)
```

In display mode, this widget will evaluate the expression and display the count of the number snapshot files in the database.

## Example 2

For example, enter the following absolute TACTIC Expression as the value for the Expression Value Element widget:

```
@COUNT(sthpw/login.sthpw/login)
```

In display mode, this widget will evaluate the expression and display the count of the number of logins in the login table.

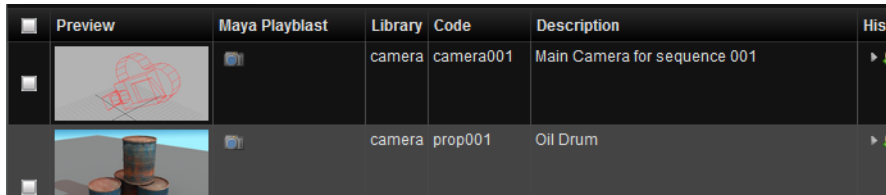
## Example 3

For example, enter the following absolute TACTIC Expression as the value for the Expression Value Element widget:

```
@GET(sthpw/task["context = 'model'"].code)
```

In display mode, this widget will evaluate the expression and display the code for all the tasks where the context is 'model'.

## Button



## Description

The ButtonElementWdg is an icon that behaves as a button which must be placed inside a TableLayoutWdg. This widget allows you to add a simple button with in a table that when pressed, will execute a JavaScript callback. This callback can be inline to the element definition or it can point to a script code or script path in the Script Editor ( "custom\_scripts" table).

## Info

<b>Name</b>	Button
<b>Class</b>	tactic.ui.table.ButtonElementWdg
<b>Category</b>	Button
<b>Supported Interfaces</b>	TableLayoutWdg
<b>TACTIC Version Support</b>	3.0 +
<b>Required database columns</b>	none

## Usage

The usage of the button widget is very simple. It appears as an icon in the cell of a table. Hovering over it will light the color and if a "tool\_tip" is defined, then this message will appear. Clicking on the button will activate the callback and perform whatever action has been set up for this button.

## Implementation

The button widget is often implemented through the view options when creating/editing a column.

### Options

<b>icon</b>	The icon to display for the button. See IconWdg for reference.
<b>script</b>	Points to the script code that is executed when the button is clicked
<b>path</b>	Points to the folder/script_name of the config/custom_script that is to be executed when the button is clicked (This is recommended option)
<b>icon_tip</b>	Text to display as a tool-tip when mouse is hovering over icon
<b>enable</b>	Expression to determine whether the button is enabled or not
<b>cbjs_action</b>	Inline script

## Advanced

Simple inline alert using default icons.

```
<element name='my_button' title='My Button Example'>
  <display class='tactic.ui.table.ButtonElementWdg'>
    <cbjs_action>alert('Pressed')</cbjs_action>
  </display>
</element>
```

Add a custom icon listed in IconWdg

```
<element name='my_button' title='My Button Example'>
  <display class='tactic.ui.table.ButtonElementWdg'>
    <icon>FILM</icon>
    <cbjs_action>alert('Pressed')</cbjs_action>
  </display>
</element>
```

A more complex inline code using CDATA

```
<element name='my_button' title='My Button Example'>
  <display class='tactic.ui.table.ButtonElementWdg'>
    <icon>FILM</icon>
    <cbjs_action>
<![CDATA[
if ( confirm("Are you sure you want to press") ){
  alert("do something...")
}
]]>
    </cbjs_action>
  </display>
</element>
```

This will call the custom\_script with the folder "test" and title "my\_test\_script". Inlining long JavaScript can be messy and this provides a means of separating out the behavior logic from the configuration. The script that is called will have a bvr object available to it. This behavior object will have a number of attributes which are useful for extracting information about the button that was pressed.

```
<element name='my_button' title='My Button Example'>
  <display class='tactic.ui.table.ButtonElementWdg'>
    <icon>FILM</icon>
    <path>test/my_test_script</path>
  </display>
</element>
```

## Link Element

<input type="checkbox"/>	Preview	Category	Asset Cod	link	Title
<input type="checkbox"/>		video	asset001		Nuclear Explosion
<input type="checkbox"/>		video	asset002	<a href="http://support.southpawtech.com">http://support.southpawtech.com</a>	Car Chase
<input type="checkbox"/>		3d	asset003		Car Warehouse

## Description

The Link Element Widget facilitates creation of a hyperlink. Clicking on the link button opens the hyperlink in a new tab in the web browser.

## Info

<b>Name</b>	Link Element
<b>Common Title</b>	Link
<b>Class</b>	Link
<b>TACTIC Version Support</b>	3.0.0 +
<b>Required database columns</b>	none

## Usage

Go into edit mode for the Link column. Specify the full URL to a hyperlink, such as: <http://support.southpawtech.com>.

Save the data and refresh the view.

Click on the link icon and the link to the web page will be opened in a new tab.

## Implementation

The Link Element Widget can be created using the Create New Column and specifying: Display -> Widget -> **Link**.

## Options

The ability to specify a customize icon to appears in the row.

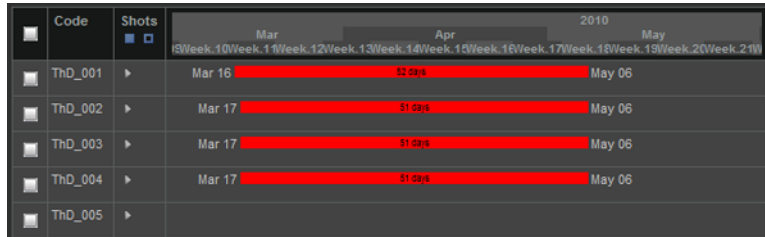
## Advanced

```
<element name="link" title="link" edit="true" color="false">
  <display widget="link"/>
</element>
```

# Gantt

## Description

The Gantt widget has the capability of displaying all projects schedules along with sequences and tasks schedules. With the widget you can switch between weeks to months view. This widget can be utilized and edited in multiple different ways. It also displays the start and end date along with the amount of days.



## Info

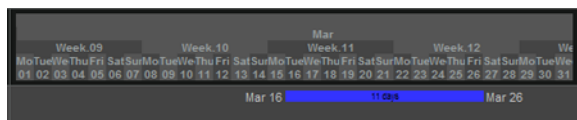
<b>Name</b>	Calendar Gantt Widget
<b>Class</b>	GanttWdg
<b>Category</b>	Common Columns
<b>Supported Interfaces</b>	
<b>TACTIC Version Support</b>	2.6.0+
<b>Required database columns</b>	configurable.

## Usage

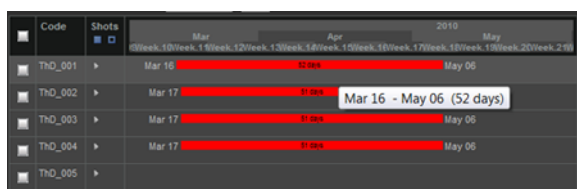
There are many ways to edit the Gantt Widget. You can also edit what part of the month, week or year of the schedule to view. Clicking on the header date of the Gantt Widget will toggle the different viewing options.



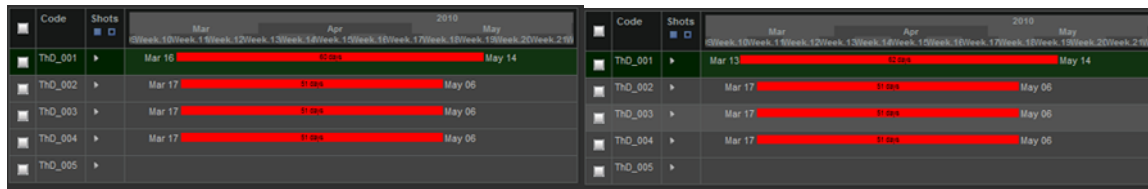
Above shows two different displays of viewing the range of the date. Clicking on the weeks will toggle to another viewing range.



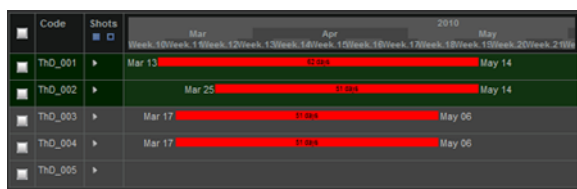
The bars that show the schedule can also be edited using the UI. Hovering the mouse over the bars will popup a window that will display the dates of the schedule.



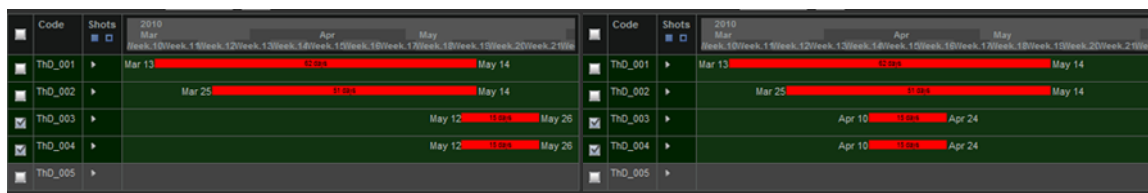
The bars can also be edited by selecting the start and end dates and sliding the either end from the right to left. The first image below shows the end of the date stretched to May 14 and the second image shows the start date stretched back to March 13.



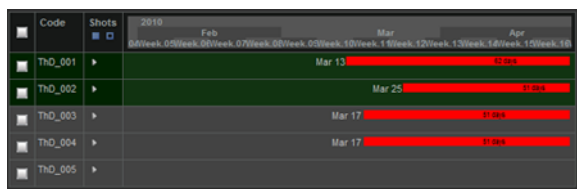
The schedule bar can also move sideways while keeping the number of days constant by selecting the bar and shifting it from left to right.



The Gantt Widget can also be edited using multi selection. Whether it is changing the end date, start date or sliding the bars forward and backward, the Gantt Widget can handle it. Below are images of a few examples of having the sequences multi-selected and edited.

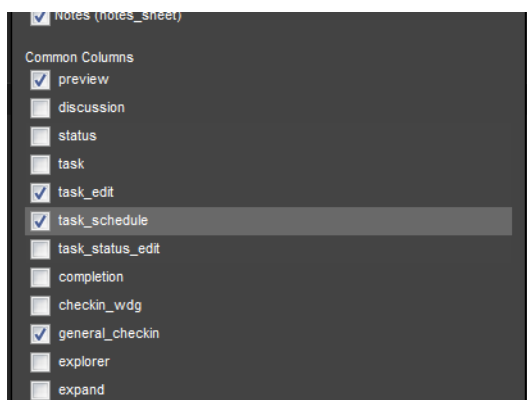


The Gantt Widget also has the capability of sliding the full time line by selecting the empty area of the widget and dragging the mouse left or right.



The Gantt Widget can be found under the column manager as task schedule.





## Advanced

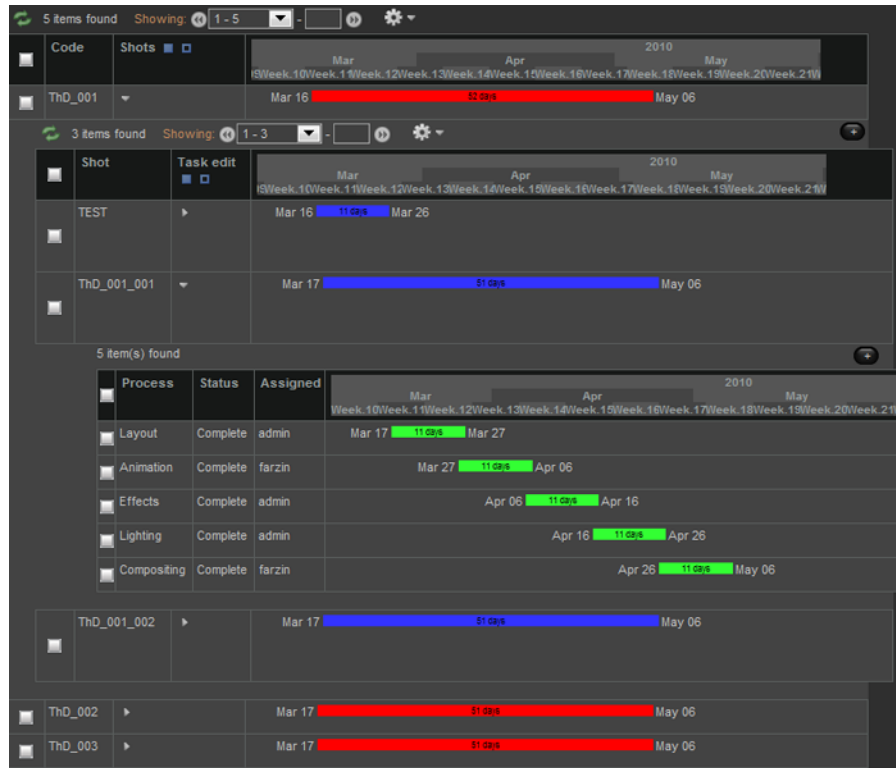
The following example illustrates a Gantt Widget that shows all tasks for a project, the schedule for all asset tasks, and the schedule for all shot tasks.

```
<element name='task_schedule'>
  <display class='tactic.ui.table.GanttElementWdg'>
    <options>[
      {
        "start_date_expr": "@MIN(sthpw/task.bid_start_date)",
        "end_date_expr": "@MAX(sthpw/task.bid_end_date)",
        "color": "white",
        "edit": "true",
        "default": "true"
      },
      {
        "start_date_expr": "@MIN(sthpw/task['search_type', '~', 'asset'].bid_start_date)",
        "end_date_expr": "@MAX(sthpw/task['search_type', '~', 'asset'].bid_end_date)",
        "color": "red",
        "edit": "true",
        "default": "false"
      },
      {
        "start_date_expr": "@MIN(sthpw/task['search_type', '~', 'shot'].bid_start_date)",
        "end_date_expr": "@MAX(sthpw/task['search_type', '~', 'shot'].bid_end_date)",
        "color": "blue",
        "edit": "true",
        "default": "false"
      }
    ]</options>
  </display>
  <action class='tactic.ui.table.GanttCbK'>
    <subjects>@SOBJECT(prod/shot.sthpw/task)</subjects>
    <options>[
      {
        "prefix": "bid",
        "subjects": "@SOBJECT(sthpw/task)",
        "mode": "cascade"
      },
      {
        "prefix": "bid",
        "subjects": "@SOBJECT(sthpw/task['search_type', '~', 'asset'])",
        "mode": "cascade"
      },
      {
        "prefix": "bid",
        "subjects": "@SOBJECT(sthpw/task['search_type', '~', 'shot'])",
        "mode": "cascade"
      }
    ]</options>
  </action>
</element>
```

```

    }
  ]</options>
</action>
</element>

```



Note: There are 3 editable bars in the display options in the above example and therefore, there are 3 corresponding action options. The 'prefix' action option assumes that the column in the table is named like <prefix>\_start\_date and <prefix>\_end\_date. If your column names are different, you would want to use the action\_option "start\_date\_col" and "end\_date\_col" with the full column name as the value.

# Hidden Row

## Description

The HiddenRowToggleWdg is used to add a cell to a table which when toggled, exposes a hidden view. This view supports the embedding the following Widgets:

- TableLayoutWdg
- CustomLayoutWdg
- ViewPanelWdg

The screenshot displays the TACTIC interface. At the top, there's a header bar with various icons and a search bar. Below it, a table lists items with columns for 'Wor', 'Process', 'Description', 'Milestone', 'Status', 'Assigned', 'Supervisor', and 'Prior'. The table has four rows: 'design', 'rough', 'final', and 'delivery'. Each row has a toggle arrow in the 'Status' column. To the right of the table is a calendar view for August 2011, showing dates from Wed 27 to Mon 08. The calendar highlights specific dates corresponding to the tasks: Jul 28 for 'design', Aug 03 for 'rough', Aug 02 for 'final', and Aug 04 for 'delivery'.

## Info

<b>Name</b>	HiddenRowToggleWdg
<b>Class</b>	tactic.ui.table.HiddenRowToggleWdg
<b>Category</b>	Common Columns
<b>Supported Interfaces</b>	TableWdg
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	none

## Usage

The HiddenRowToggleWdg is primarily a configuration tool which provides very simple usage for the user. By clicking the expand arrow, the hidden row will expand. Also, to batch expand the same HiddenRow for multiple rows in the table, select the desired rows (SObjects) and in the table header, do one of the following:

- Click the triangle to expand or collapse the HiddenRow for the selected SObjects.

## Options

<b>dynamic_class</b>	The class to embed in the hidden row
<b>new</b>	Deprecated

<b>dynamic</b>	Deprecated
<b>static</b>	The view is loaded when the page loads
<b>parent_key</b>	The parent key of the parent SObject (Internal)

## Advanced

The following HiddenRowToggleWdg is defined in the **definition** view for a prod/sequence. The embedded table shows a view of prod/shot SObjects in a view called **shot\_hierarchy**

```
<element name='shots'>
  <display class='HiddenRowToggleWdg'>
    <dynamic_class>tactic.ui.panel.TableLayoutWdg</dynamic_class>
    <search_type>prod/shot</search_type>
    <view>shot_hierarchy</view>
    <mode>simple</mode>
    <do_search>true</do_search>
    <show_row_select>>false</show_row_select>
  </display>
</element>
```

The following HiddenRowToggleWdg is used to show a view of prod/asset SObjects which have been planned (associated) to a prod/shot SObject. In this case, the available **<expression/>** option is used in the TableLayoutWdg to get the assets by traversing through the following search types:

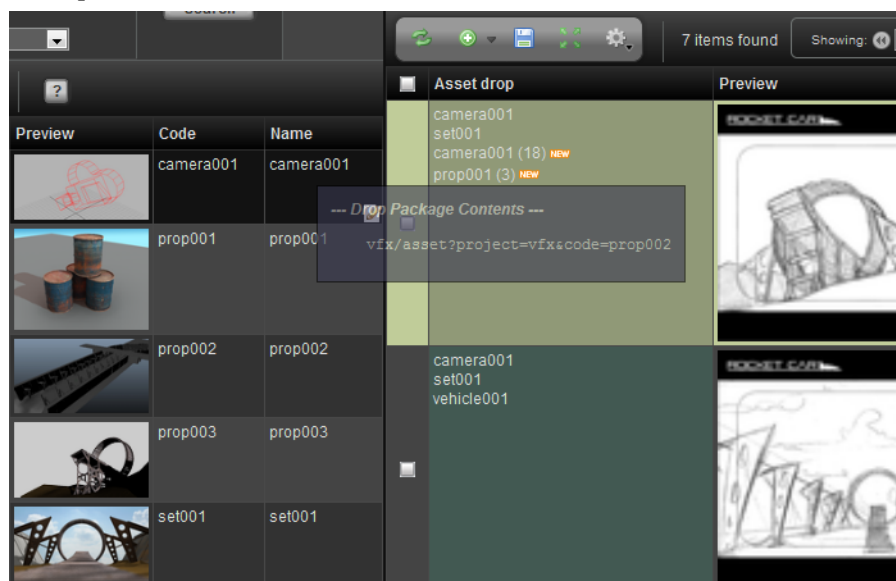
**prod/shot -> prod/shot\_instance -> prod/asset**

```
<element name='assets'>
  <display class='HiddenRowToggleWdg'>
    <dynamic_class>tactic.ui.panel.TableLayoutWdg</dynamic_class>
    <search_type>prod/asset</search_type>
    <view>assets_hierarchy</view>
    <expression>@SOBJECT(prod/shot_instance.prod/asset)</expression>
    <mode>simple</mode>
    <do_search>true</do_search>
    <show_row_select>>false</show_row_select>
  </display>
</element>
```

The following example shows how the dynamic\_class is used to point to which widget to use in the hidden row.

```
<element name='tasks'>
  <display class='HiddenRowToggleWdg'>
    <dynamic_class>tactic.ui.panel.TableLayoutWdg</dynamic_class>
    <search_type>sthpw/task</search_type>
    <view>task_hierarchy</view>
    <mode/>
    <do_search>true</do_search>
    <show_row_select>true</show_row_select>
  </display>
</element>
```

## Drop Item



## Description

Facilitates drag-and-drop of an item between 2 views. For example, drag a user from one view and drop it into a user group.

## Info

<b>Name</b>	Drop Element Widget
<b>Common Title</b>	Drop Element Widget
<b>Class</b>	tactic.ui.table.DropElementWdg
<b>TACTIC Version Support</b>	3.0.0 +
<b>Required database columns</b>	none

## Usage

For example, in the Shot Planner view, individual assets can be added to a shot by simply dragging the asset from one view and dropping it onto the shot in another view. Once the asset is dropped onto the shot, the asset will appear in the column with a "NEW" flag. Hit the save button in the shot view to preserve the changes.

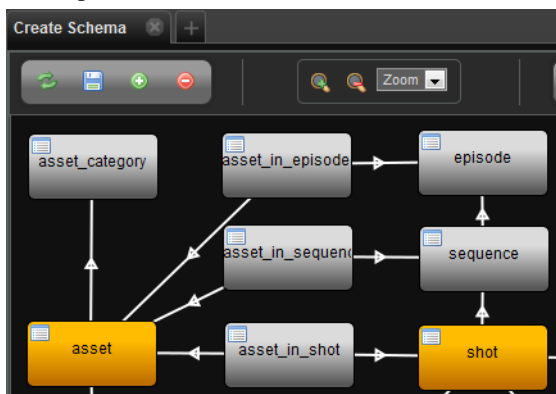
## Options

<b>Accepted Drop Type</b>	The acceptable sType that can be clicked on to be dragged and dropped onto another type. For example, sType is vfx/asset.
<b>Instance Type</b>	For the item that is being dragged, it is the sType that the item can be dropped onto. For example, sType is vfx/asset_in_shot"
<b>Chjs Drop Action</b>	The call back JavaScript to run each time an item is dropped into the column.
<b>Display Expr</b>	The expression to run to display in view mode. For example "@"

## Implementation

A many-to-many relationship between the 2 types needs to be created in the Schema Editor. By convention, the "join" node that need to be created to connect the 2 types should be named: "<sType1>\_in\_<sType2> ". For example, for the join node named: "asset\_in\_shot". The "asset\_in\_shot" node stores the data representing the relationship between the asset and which shot it appears in.

The view where the item to be dropped onto the Drop Element column of, can exist in a custom layout table or a view opened in a new window within the TACTIC session.



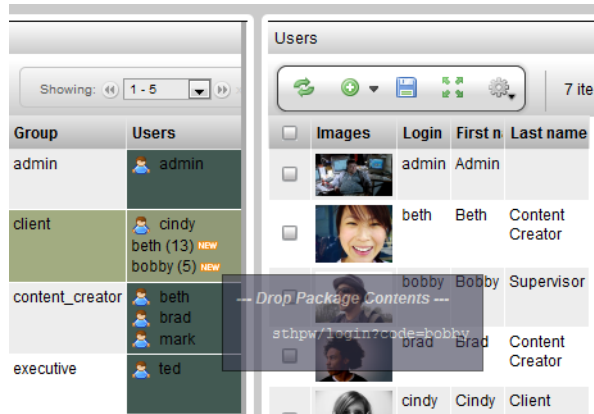
```
<element name="asset_drop" width="333px" edit="false">
  <display class="tactic.ui.table.DropElementWdg">
    <instance_type>vfx/asset_in_shot</instance_type>
    <accepted_drop_type>vfx/asset</accepted_drop_type>
    <css_background-color>#425952</css_background-color>
  </display>
  <action class="tactic.ui.table.DropElementAction">
    <instance_type>vfx/asset_in_shot</instance_type>
  </action>
</element>
```

## Examples

Example 1: implementation of "asset\_in\_shot", where an asset can be drag and dropped onto a shot:

```
<element name="asset_drop" width="333px" edit="false">
  <display class="tactic.ui.table.DropElementWdg">
    <instance_type>vfx/asset_in_shot</instance_type>
    <accepted_drop_type>vfx/asset</accepted_drop_type>
    <css_background-color>#425952</css_background-color>
  </display>
  <action class="tactic.ui.table.DropElementAction">
    <instance_type>vfx/asset_in_shot</instance_type>
  </action>
</element>
```

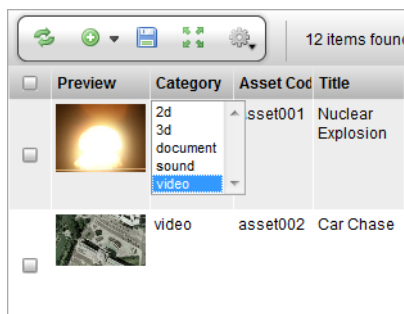
Example 2: implementation of "user\_in\_group", where a user can be drag and dropped onto a group:



```
<element name="users">
  <display class="tactic.ui.table.DropElementWdg">
    <css_background-color>#425952</css_background-color>
    <instance_type>sthpw/login_in_group</instance_type>
    <accepted_drop_type>sthpw/login</accepted_drop_type>
  </display>
</element>
```

# Edit Widgets

## Select



## Description

The Select Widget is a simple widget version of an HTML drop down selection box. The widget is used for making a selection from a predefined list of items. Many built-in dropdown widgets in TACTIC extend from the Select Widget.

## Info

<b>Name</b>	Select Widget
<b>Class</b>	SelectWdg
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	None, but typically this is attached to a data column

## Usage

Usage of the Select Widget is straightforward. Simply click on the Select Widget button to open the drop down selection box. Then, select one of the menu items. Sometimes items are grouped and separated by a group label represented as << label >>. In that case, selecting the group label will trigger a warning pop-up. To unset a value, you can usually select the empty value with the label '-- Select --'.

## Implementation

The select is often setup in the Edit Column definition -> Edit Tab. It is edited for the state for column data where the user should only be able to choose from a list of predefined values.

## Options

<b>values</b>	A list of data values separated by the pipe character ' ', e.g. model anim lighting
<b>labels</b>	A list of display labels separated by the pipe character ' ', e.g. Model Anim LGT
<b>empty</b>	When set to true, the Select Widget will contain an empty option.
<b>values_expr</b>	This serves the same purpose as values but in the form of an expression. The input item of the expression has to exist for this to function properly.(ie @GET(vfx/sequence.code) ). If it is used in the menu of an item in insert mode, you should set mode_expr to 'absolute'
<b>labels_expr</b>	This serves the same purpose as labels but in the form of an expression. The input item of the expression has to exist for this to function properly (ie @GET(vfx/



	sequence.name) ). If it is used in the menu of an item in insert mode, you should set mode_expr to 'absolute'
<b>mode_expr</b>	If left unset, the default is to use the current item in the expression defined in values_expr and labels_expr. If set to 'absolute', the input item for the expression will be an empty list.
<b>query</b>	In the form of <search_type> <value> <label>, you can instruct the widget to retrieve the values and labels from the a particular sType. For example, to get all the asset codes from the sType 'vfx/asset', you can use 'vfx/asset code code'. To change the label to display asset's name instead, you can use 'vfx/asset code name'.

## Advanced

The following example uses the query option to get the **code** of a parent shot item but, display the **name** value in the list. This query option is older. The values\_expr and labels\_expr option are preferred.

```
<element name="parent_code">
  <display class="SelectWdg">
    <query>prod/shot|code|name</query>
  </display>
</element>
```

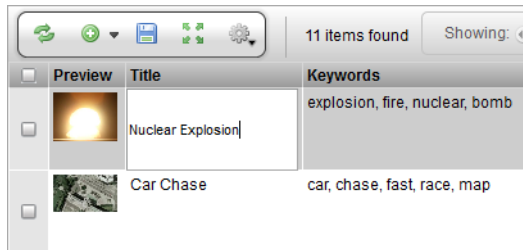
The following gets the same result but, uses expressions. This allows for more robust queries for values and labels.

```
<element name="parent_code">
  <display class="SelectWdg">
    <mode_expr>absolute</mode_expr>
    <values_expr>@GET(prod/shot.code)</values_expr>
    <labels_expr>@GET(prod/shot.name)</labels_expr>
  </display>
</element>
```

The following sets a hard coded list of values and labels for the SelectWdg.

```
<element name="status">
  <display class="SelectWdg">
    <values>waiting|in_progress|complete</values>
    <labels>Waiting|In Progress|Complete</labels>
  </display>
</element>
```

## Text Input



## Description

The TextWdg is a basic form element in which a single line of text can be entered. (To enter multiple lines, use the TextAreaWdg instead.) It maps directly to the HTML text input. It can be used independently or as an edit element in the TableLayoutWdg or EditWdg.

## Info

<b>Name</b>	Text Input
<b>Class</b>	pyasm.widget.TextWdg
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	none

## Implementation

Basic example of a typical usage of a TextWdg

## Options

<b>size</b>	Determine the width of the text widget. Default is "50".
<b>read_only</b>	true/false - determines whether the widget can have its text contents altered.

## Advanced

Simple example which displays text widget that is fully editable:

```
<element name='first_name'>
  <display class='pyasm.widget.TextWdg' />
</element>
```

A text widget that only allows integer input. The size is reduced to 5.

```
<element name='age'>
  <display class='pyasm.widget.TextWdg'>
    <size>5</size>
  </display>
</element>
```

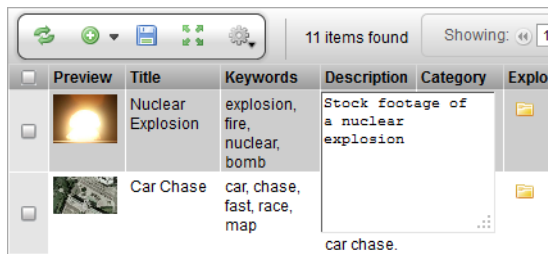
A simple example of the TextWdg in Python:

```
from pyasm.widget import TextWdg

div = DivWdg()
text_wdg = TextWdg("age")
```

```
text_wdg.set_option("size", "20")  
div.add(text_wdg)
```

## Text Area



## Description

The TextAreaWdg is a simple text widget which is used for editing full-text. The widget supports using the ENTER key for adding new lines (the ENTER key is often not supported on text entry widgets where CTRL+ENTER is used.) This widget can also be configured to display a larger canvas to work on.

## Info

<b>Name</b>	TextAreaWdg
<b>Class</b>	pyasm.widget.TextAreaWdg
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	requires a database column for storing the text data.

## Implementation

The TextAreaWdg is used in Edit scenarios where full text input is required. There is control for the columns (characters across) and rows (characters down).

## Options

<b>cols</b>	The number of character columns in the TextArea
<b>rows</b>	The number of character rows in the TextArea

## Advanced

The following example is a default implementation. The default number of cols is **50** and the default number of rows is **3**.

```
<element name="subject">
  <display class="TextAreaWdg"/>
</element>
```

The following example creates a large text area which could be used for writing large amounts of full-text.

```
<element name="summary">
  <display class="TextAreaWdg">
    <cols>100</cols>
    <rows>30</rows>
  </display>
</element>
```

## Calendar Input Widget

Assigned	Bid start date	Bid end date	Jul	
ted	Jul 19, 2011	Jul 25, 2011	Jul 19	Jul 25
ted	Jul 20, 2011	Aug 05, 2011	Jul 20	Aug 05
ted			Jul 28	Aug 03
bobby			Aug 01	Aug 05
mark			Aug 06	Aug
admin			Aug 10	
cindy				
ted				
beth				
beth	Sep 04, 2011	Sep 15, 2011		
ted	Sep 13, 2011	Sep 14, 2011		

## Description

The CalendarInputWdg displays a navigable calendar where dates can be selected. It is an input widget that conforms to the BaseInputWdg interface and is used for inline editing or as one of the items in the EditWdg layout.

January, 2012						
Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

## Info

<b>Name</b>	Calendar Input
<b>Class</b>	tactic.ui.widget.CalendarInputWdg
<b>Category</b>	Input widget
<b>Supported Interfaces</b>	EditWdg, TableLayoutWdg (edit view)
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	none unless editing a specific date column

## Implementation

The simple implementation does not require any options. It displays a non-editable text box with a value that represents a date. Clicking on the cell opens up the calendar widget.

## Options

<b>first_day_of_week</b>	Integer representing first day of the week (0=Sunday, 6=Saturday)
<b>read_only</b>	Sets the widget to be read only. In read-only mode, clicking on the cell does not bring up the calendar for input. Only a text box with the date value is displayed.

## Advanced

The simplest and most common usage is the default implementation.

```
<element name='start_date'>
```

```
<display class='tactic.ui.widget.CalendarInputWdg' />
</element>
```

To set the work week to start on a different day than Sunday, change the `first_day_of_week` . This option is an integer which represents the days of the week where 0=Sunday and 6=Saturday.

```
<element name='start_date'>
  <display class='tactic.ui.widget.CalendarInputWdg'>
    <first_day_of_week>6</first_day_of_week>
  </display>
</element>
```

# Common Widgets

## Completion

Process	Status	Completion	Assigned
ingest	approved	<div><div></div></div> 100.0%	brad
roto	in_progress	<div><div></div></div> 40.0%	beth
depth	revise	<div><div></div></div> 60.0%	brad
comp	ready	<div><div></div></div> 20.0%	mark
deliver	review	<div><div></div></div> 80.0%	mark
ingest	revise	<div><div></div></div> 60.0%	brad
roto	in_progress	<div><div></div></div> 40.0%	beth
depth	review	<div><div></div></div> 80.0%	beth
comp	revise	<div><div></div></div> 60.0%	brad
deliver	ready	<div><div></div></div> 20.0%	brad
		Total	
		<div><div></div></div> 24.0%	

## Description

The Task Completion Widget provides a graphical bar chart to represent the progress of an item by the completion rate of its child tasks.

## Info

<b>Name</b>	Task Completion Widget
<b>Common Title</b>	Task Completion Widget
<b>Class</b>	tactic.ui.table.TaskCompletionWdg
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	none

## Usage

This is a display-only widget. If all the tasks are completed for a shot, the bar reading would be 100%. Otherwise, a partial completion would be calculated based on tallying all the child tasks. If there are no tasks for the item, "No tasks" is displayed.

Total Tasks	Tasks	Completion
	▶	No tasks
24	▶	<div><div></div></div> 58.0%
16	▶	<div><div></div></div> 25.0%
		Total
		<div><div></div></div> 41.5%

## Implementation

It is a common column that can be added using the Column Manager. The item name is "completion".

## Options

task_expr	An expression to get to the tasks relative to the current sObject. e.g. @SOBJECT(prod/shot.sthpw/task)
-----------	--

## Advanced

```
<element name="completion" edit="false">
  <display class="tactic.ui.table.TaskCompletionWdg" />
</element>
```

## How The "Completion" is Calculated

### Example 1:

Let's say that we have a task in a pipeline with the following processes:

4 processes: Design, Rough, Modeling, Delivery

Let's say that for each process, there are:

4 statuses: Unassigned, In Progress, Ready\_for\_Review, Approved.

If the task is in the status: Unassigned, the task is 0% complete.

If the task is in the status: Started, the task is 33.3% complete.

If the task is in the status: Ready\_for\_Review , the task is 66.6% complete.

If the task is in the status: Approved, the task is 100% complete.

Let's say the task in the Rough process has the status 'Approved'. That means that is 100% complete for the Rough process. In the other 3 processes, it is at Unassigned, which is 0% complete.

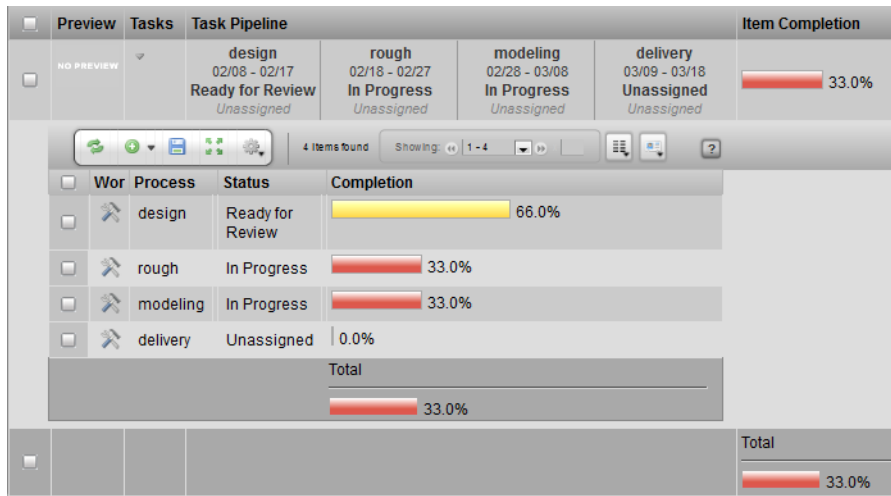
Preview	Tasks	Task Pipeline				Item Completion
<input type="checkbox"/>	NO PREVIEW	design 02/08 - 02/17 Approved Unassigned	rough 02/18 - 02/27 Unassigned Unassigned	modeling 02/28 - 03/08 Unassigned Unassigned	delivery 03/09 - 03/18 Unassigned Unassigned	<div><div></div></div> 25.0%
4 items found Showing: 1 - 4						
<input type="checkbox"/>	Wor	Process	Status	Completion		
<input type="checkbox"/>		design	Approved	<div><div></div></div>	100.0%	
<input type="checkbox"/>		rough	Unassigned	<div><div></div></div>	0.0%	
<input type="checkbox"/>		modeling	Unassigned	<div><div></div></div>	0.0%	
<input type="checkbox"/>		delivery	Unassigned	<div><div></div></div>	0.0%	
Total				<div><div></div></div>	25.0%	
Total						<div><div></div></div> 25.0%

Then, the TOTAL completion would be  $(1.0 + 0 + 0 + 0) / 4 = 25\%$  complete.

### Example 2:

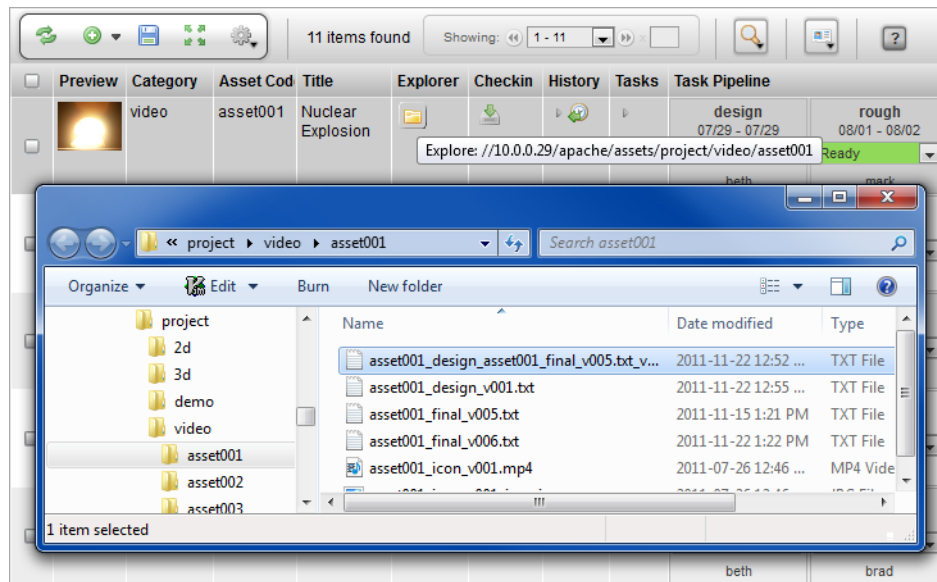


Using the same process and task statuses, let's say the task in the Rough process has the status 'Ready\_for\_Review'. That means that is 66% complete for the Rough process. In the other 2 processes, it is at Started, which is 33.3% complete. In the last processes, it is at Unassigned, which is 0% complete.



Then, the TOTAL completion would be  $(0.666 + .333 + .333 + 0) / 4 = 33.3\%$  complete. .

## Explorer Button



## Description

The Explorer Widget can be configured to launch Windows Explorer for Windows (or Finder for OSX). It can be configured to open to a directory which is either the sandbox or to the repository of the corresponding item.

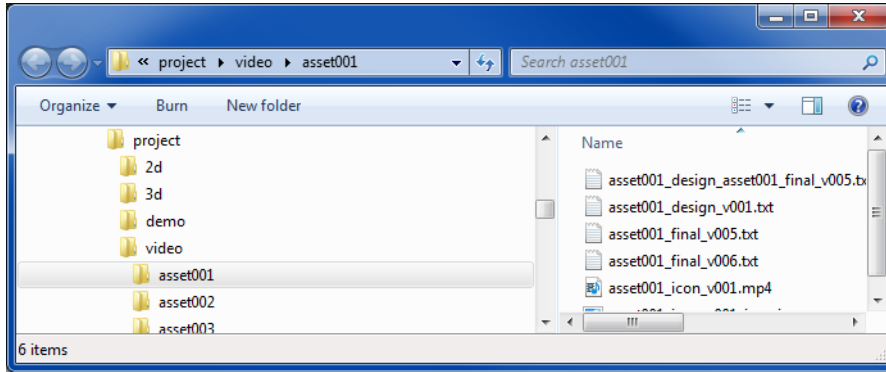
## Info

<b>Name</b>	Explorer
<b>Class</b>	tactic.ui.table.ExplorerElementWdg
<b>Category</b>	Common Columns
<b>TACTIC Version Support</b>	3.0+
<b>Required database columns</b>	none

## Implementation

When added to the view, the Explorer Widget button is represented as an icon of a folder. The button opens up Windows Explorer (or Finder on OSX). This gives the user a quick starting point for navigating to a directory that is relevant to the corresponding item. The convenience is greater when the repository contains a lot of items or the directory folder structure is very deep. Users save time by not having to navigate through endless directories to get to where they need to go to do work.

By default, the Explore Widget opens a window to the corresponding item if it exists in the user's sandbox.



## Options

<b>mode</b>	sandbox client_repo - determines what directory to go to when the explorer button is pressed.
-------------	---

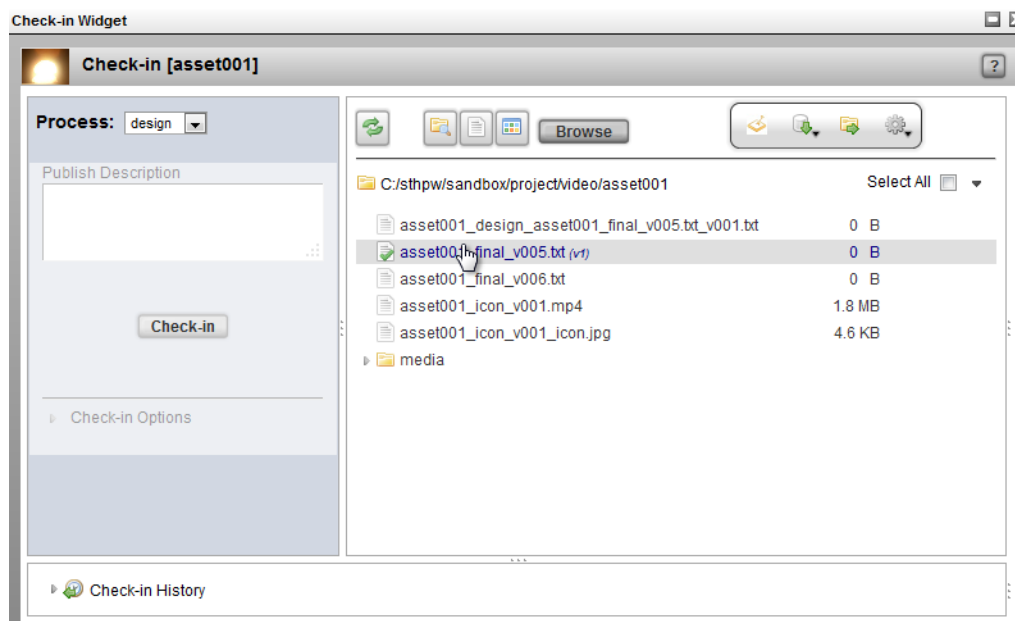
## Advanced

The following example configures the Explorer Widget to browse to the assets directory as specified in the **Tactic Config File**

```
<element name='explorer'>
  <display class='tactic.ui.table.ExplorerElementWdg'>
    <mode>client_repo</mode>
  </display>
</element>
```

```
<element name='explorer'>
  <display class='tactic.ui.table.ExplorerElementWdg' />
</element>
```

## General Check-in Widget



## Description

This is the new preferred Check-in Widget for 3.7+. It makes use of the Java Applet to accomplish various kinds of check-in functions like checking in a single file, sequences of files, or directories. The **copy** or **preallocate** transfer mode should be used when dealing with a large file transfer. **Upload** transfer mode only supports checking in single files and sequences of files. **Upload** is set to be the default in case new users do not have the handoff directory readily set up.

## Info

<b>Name</b>	General Check-in Widget
<b>Class</b>	tactic.ui.widget.CheckinWdg
<b>Category</b>	Widget
<b>Supported Interfaces</b>	TableLayoutWdg
<b>TACTIC Version Support</b>	3.7.0 +
<b>Required database columns</b>	none

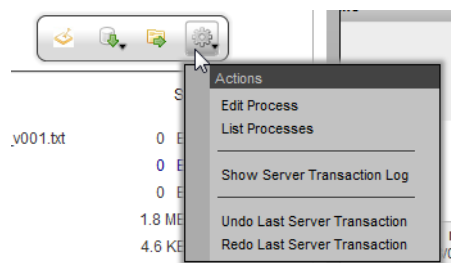
## Options

<b>transfer_mode</b>	<b>upload</b> , <b>copy</b> and <b>move</b> are supported. <b>copy</b> is recommended for most situations when users are usually granted only read access to the TACTIC asset repo. ( <i>default is copy</i> )
<b>mode</b>	<b>sequence</b> , <b>file</b> , <b>dir</b> , and <b>add</b> are supported. <b>sequence</b> is for file sequence checkin; <b>file</b> is for single file checkin; <b>dir</b> is for directory checkin; and <b>add</b> is for appending file or dir to an existing snapshot. If not specified, multiple selections will be available for the user to choose. Note: upload transfer mode only supports single file or file sequence checkin.

<b>checkin_script_path</b>	a custom checkin script path to specify an override on what functions get called during a checkin. Note: If trying to do some preprocessing with the file or directory before checking in, just make use of validate_script_path function using Client Trigger. Client Trigger works by setting up this check-in script as a Client Trigger callback that affects the search type rather than just a column definition.
<b>validate_script_path</b>	a script path pointing to a JavaScript file that is run before the actual checkin. If it throws an error using "throw(<error message>)", the checkin will not initiate. This path can also use it to run some client-side preprocessing of the file or directory. It is not set up as a display option but rather as a Client Trigger callback.
<b>checkout_script_path</b>	a custom check-out script path to specify to override what happens during a check-out.
<b>process</b>	If set, the process specified will be pre-selected when the General Check-in Widget is drawn,
<b>lock_process</b>	If set to true, the user will not be able to choose a different process during a checkin, in the General Check-in Widget
<b>show_context</b>	When set to true, the <b>context</b> will be displayed to the user. ( <i>default is false</i> )

## Gear Menu Options

The Gear Menu in the Check-in Widget provides the following administration options:



<b>Edit Process</b>	Load the process options pop-up. <i>The process and subcontext options are described further in the sections below</i>
<b>List Processes</b>	List all of the processes for the current pipeline. This provides the same access to the as the Edit Process option but for all processes.
<b>Show Server Transaction Log</b>	Show the standard server transaction log
<b>Undo Last Server Transaction</b>	Undo the last transaction. <i>When undoing a checkin, the files will also be removed in the file system.</i>
<b>Redo Last Server Transaction</b>	Redo the last transaction. <i>When redoing a checkin, the files will be restored in the file system.</i>

## Implementation

The default settings will allow a user to check in files to an assets in the "publish" process. It provides a very general and loosely enforced workflow to check in and manage files. Often, it is required, that a particular process has very strict enforcement of naming conventions and check-in procedures.

The General Check-in widget is highly configured and can be tuned precisely for each part of the process. The various customizations can fall into the following categories:

Validation  
Subcontext options  
Custom interface  
Custom check-in script  
Naming conventions

Each of these can be customized for the particular widget or at the process level.

## Validation

Validation is a custom script that will be run before the check-in process occurs. It provides the ability to check that all files in the checkin conform to some custom logic required for a successful checkin. If the validation script fails, then the entire checkin is aborted.

## Client Side Triggers

A client trigger set up allows control over what check-in script or validate\_checkin script to call during a checkin. Here is an example of how to set the checkin/validate\_folder script to run before the check in of prod/asset. The event name is CheckinWdg|validate\_script\_path|<search\_type>. If only a particular process is desired to be run on check in for, like "texture", the event name would become CheckinWdg|validate\_script\_path|prod/asset|texture. To override the checkin\_script\_path, use the event CheckinWdg|checkin\_script\_path|<search\_type>. If this event-based set-up seems a bit too involving, override the checkin\_script\_path for just this instance of the widget by using the standard display option <checkin\_script\_path>.

## Process Options

By default, the subcontext selection is set to **(auto)**. It is the simplest to use and allows TACTIC to auto generate the subcontext. Because the subcontext is auto generated, strict naming conventions for the file are often sacrificed for ease of use. By default, the checked in file will just have a version number attached to it.

It is possible to force a limited list of subcontext options on a particular checkin. This means that the files checked in will be named according to the subcontext selected and provides a limited set of approved containers in which files can be checked in.

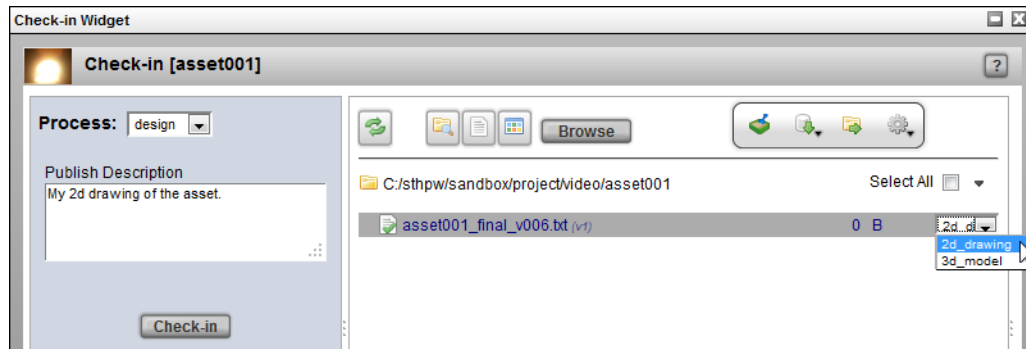
## Process/Context/Subcontext

Checkin's are always categorized by process. If there is no pipeline defined, the default process "publish" will be used. Categorizing checkin's by the process in the pipeline of an asset organizes the work done for an asset according to its product life cycle.

Another important attribute of a checkin is the context. Assets are versioned according to their context which provide a namespace for versioning checkin's of an asset. All checkin's of an asset with the same context are versioned together. The context of an asset is a particular way to view an asset.

For example, a 2D drawing of a character and a 3D model of the same character represent the same abstract asset, so are two different contents of the asset. This can be implemented in TACTIC by specifying the following in the Check-in Widget's Context Options:

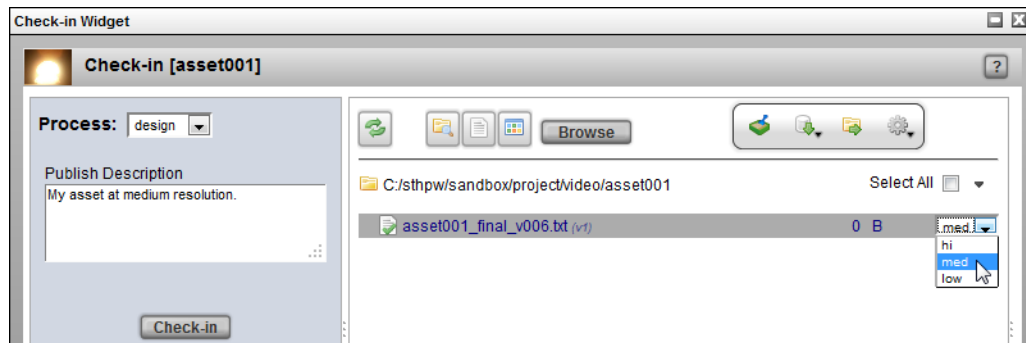
```
Context Options: 2D_drawing|3D_model
```



Although the context can be any string, most often, it is built up from other parameters. The convention usually used is "<process>/<subcontext>". All of TACTIC's built-in check-in tools assume this relationship. The subcontext provides a namespace for checking in multiple subcategories of files within a single context.

The following is an example of these subcontext options:

Subcontext Options: hi | med | low

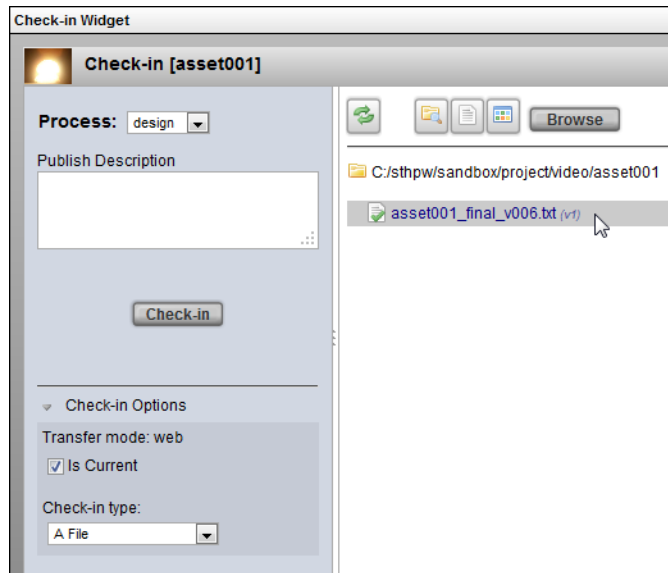


Naming conventions are often strictly enforced, meaning that the folder and the file name are automatically supplied on check in of a file to the central repository.

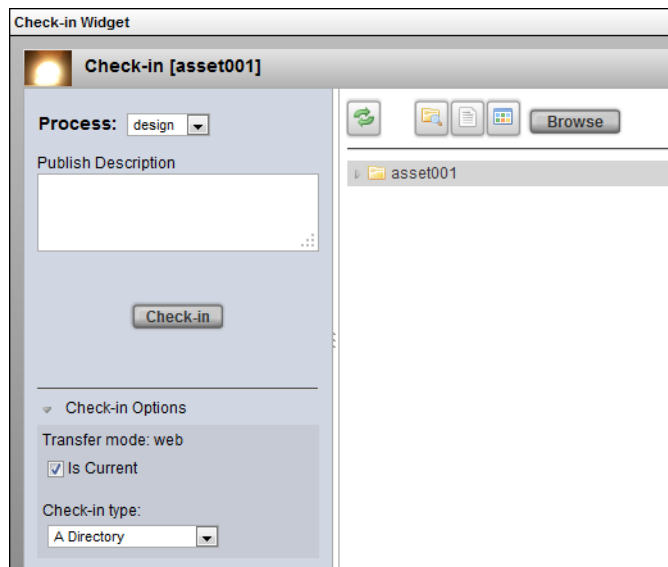
## Default Check-in Widget Options

In the panel on the right, when something from the list is selected for check in, the corresponding Check-in type (e.g., **file**, **directory**, **sequence**, **multiple individual files**) is automatically selected by the Check-in Widget.

For example, on the panel on the right, if a *file* is selected for check in, the Check-in type will automatically switch to **A File** under the Check-in Options on the bottom left:



For example, if a *folder* is selected to check in on the right panel, the Check-in type will automatically switch to **A Directory** under the Check-in Options on the bottom left:



## More Context and Subcontext Examples

### Example 1)

To check in *high resolution* and *low resolution* files for a model process, first specify the **context\_options** under:

**Checkin Widget -> Gear Menu -> Edit Process:**



Specify the following Context Options:

```
Context Options: model\hi|model\lo
```

**OR** specify the following Subcontext Options:

```
Subcontext Options: hi|lo
```

Both of the choices above give the *same* result.

Result:

```
process = model
context = model/hi    (or model/lo)
```

Only use either the context field or the subcontext field but not both fields.



## Note

If values are specified for *both* the **context\_options** and the **subcontext\_options**, only the **context\_options** will be used (the **subcontext\_options** will be ignored).

### Example 2)

To provide the same options (hi and lo) and avoid using subcontexts specify the following **context\_options**:

```
context_options: model_hi|model_lo
```

Result:

```
process = model
context = model_hi    (or model_lo)
```

Notice that the forward slash '/' was not used, which avoids using subcontexts.

### Example 3)

The following is another example of how to avoiding using subcontexts altogether.

To check in a *proxy* and a *staging* context for a model process, specify the following **context\_options**:

```
context_options: model_proxy|model_staging
```

Result:

```
process = model
context = model_proxy      (or model_staging)
```

Again, notice that the forward slash '/' was not used, which avoids using subcontexts.

## Subcontext Keywords: (auto), (main) and (text)

The following subcontext option keywords are supported:

<b>(auto)</b>	Uses the filename as the subcontext ( <i>auto is the default if no values are specified for the context or subcontext options</i> )
<b>(main)</b>	Uses the process as the context
<b>(text)</b>	Allows the user to specify their own context for the file to check in

Example for **(auto)**:

<b>process:</b>	design
<b>filename:</b>	my_checkin_file.txt
<b>subcontext option selected:</b>	(auto)

Result:

```
context = design/my_checkin_file.txt
```

Example for **(main)**:

<b>process:</b>	design
<b>subcontext option selected:</b>	(main)

Result:

```
context = design (because design is the process)
```

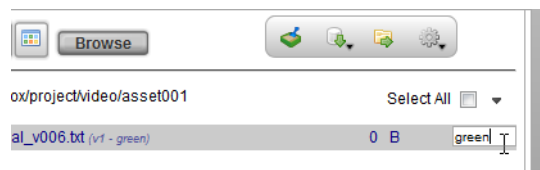
Example for **(text)**:

To check in different colors of a car for the design process eg. a green version of the car and a red version

<b>process:</b>	design
<b>subcontext option selected:</b>	(text)
<b>custom context inputted</b>	green

Result:

```
context = design/blue
```



## Providing a Custom Layout View For the Check-in Options

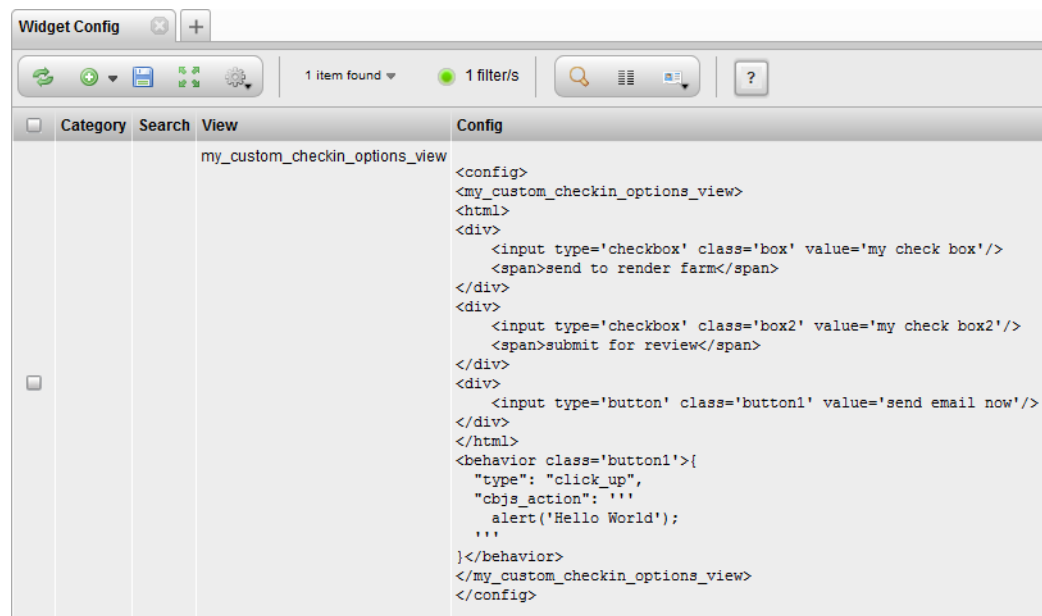
A custom layout view can be provided in the check-in panel as options.

For example, to provide check boxes during the check in to submit the job to the render farm or to submit the file for the review process, create a custom view and specify the view in the **Check Options View**.

To do this, first, create a custom view under:

### Admin Views -> Project -> Widget Config

Below is an example of a custom layout view:



note:

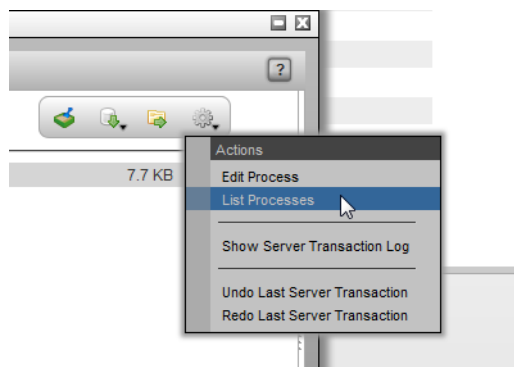
*In the example custom view above, to make use of these custom UI check boxes, more work needs to be done to override the `checkin_script` or `checkin_validate_script`.*

*The `checkin_script` and the `checkin_validate` script can be found under: Checkin Widget -> Gear Menu -> List Processes*

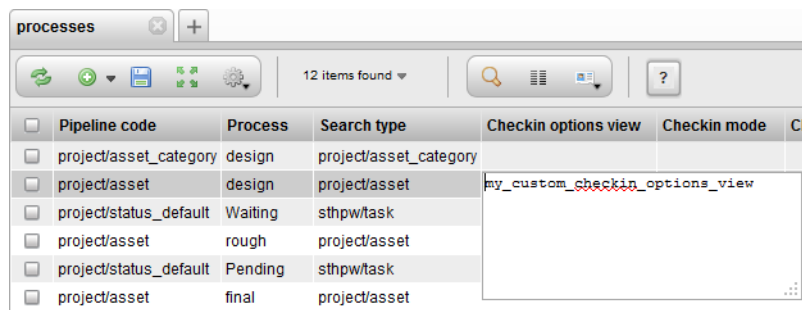
*Example validate scripts can be found at the end of this Check-in Widget doc in the section labeled Example Scripts: Example 1 and 2.*

Then, specify the name of the view under:

### Checkin Widget -> Gear Menu -> List Processes

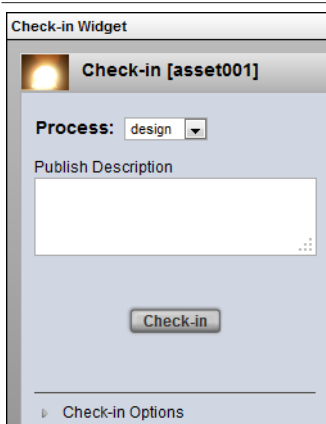


In the Check-in Options View, specify the name of the custom layout view for the check-in options:

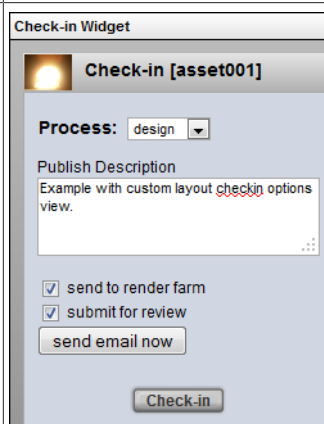


Finally, select a file to check in, the custom view with the check-in options will appear on the panel on the left.

#### Without custom check-in options:



#### With customer check-in options:



## Script Samples

This script can be saved in the Script Editor accessible through the Gear Menu.

#### Example 1: checkin/validate\_folder

```
var values = bvr.values;
var file_path = values.file_paths[0];
var sk = values.search_key;
var applet = spt.Applet.get();

var file_list = applet.list_dir(file_path);

for (var i=0; i <file_list.length; i++){

    var base =spt.path.get_basename(file_list[i]);
    if ( base == 'DATA') {
        throw('it contains a DATA folder. Checkin aborted');
    }
}
```

#### Example 2: checkin/validate\_file

```
var values = bvr.values;
```

```

var file_path = values.file_paths[0];
var sk = values.search_key;
var applet = spt.Applet.get();

var base =spt.path.get_basename(file_path);
if ( base.test(/\.\mov$/)) {
    throw('it does not have a mov extension. Validation failed.');
```

Example 3: Custom checkin\_script using display option "checkin\_script\_path". The default snapshot\_type is file, if the file extension is .mov, the snapshot\_type is set to 'mov'.

```

var file_paths = bvr.values.file_paths;
var description = bvr.values.description;
var search_key = bvr.values.search_key;
var context = bvr.values.context;
var transfer_mode = bvr.values.transfer_mode
var is_current = bvr.values.is_current;
var path = file_paths[0]
spt.app_busy.show("File Checkin", path);

var snapshot_type = 'file';
if (path.test(/\.\mov$/)){
    snapshot_type = 'mov';
}
var server = TacticServerStub.get();
snapshot = server.simple_checkin(search_key, context, path,
{description: description, mode: transfer_mode, is_current: is_current,
 snapshot_type: 'mov'});
```

## Advanced

The General Check-in Widget is usually invoked with a CheckinButtonElementWdg with a transfer mode specified. In this implementation, the process will be preselected as "texture", providing the pipeline for this sObject does contain a process named 'texture'.

```

<element name='general_checkin' title=' '>
  <display class='tactic.ui.widget.CheckinButtonElementWdg'>
    <transfer_mode>copy</transfer_mode>
    <process>texture</process>
  </display>
</element>
```

In this implementation, the process will be preselected as "model", providing the pipeline for this sObject does contain a process named 'model'. The user cannot switch to other processes in the pipeline, and only "New Directory" mode can be selected.

```

<element name='general_checkin' title=' '>
  <display class='tactic.ui.widget.CheckinButtonElementWdg'>
    <transfer_mode>copy</transfer_mode>
    <process>model</process>
    <lock_process>true</lock_process>
    <mode>dir</mode>
  </display>
</element>
```

## Checkin History

Preview	Category	Asset Cod	Title	Checkin	History	Tasks	Task Pipeline
	video	asset001	Nuclear Explosion				<div>design 07/29 - 07/29 Review beth</div> <div>rough 08/01 - 08/02 Ready mark</div> <div>final 07/26 - 07/30 Waiting beth</div> <div>delivery 08/04 - 08/04 Waiting bobby</div>

Preview	Locked	Files	Checkout	Context	Ver#	Rev#	Login	Timestamp	Description	Current	Notes sheet
				final	v6	0	beth	Nov 22, 2011 - 18:22	worked on this		
				design	v1	0	admin	Nov 22, 2011 - 17:55	Initial insert		
				final	v5	0	beth	Nov 15, 2011 - 18:21	No comment		

## Description

The Checkin History Widget is a toggle that opens a hidden row that displays all the snapshots (snapshots are checkins at a particular moment in time for a context) for an item.

## Info

Name	Checkin History Widget
Common Title	History
Class	tactic.ui.widget.SObjectCheckinHistoryWdg
Category	Common
TACTIC Version Support	3.0.0 +
Required database columns	none

## Usage

The following details are displayed by the Checkin History Widget for a task:

- **preview** of the snapshot
- whether checkout of the snapshot is **locked**
- toggle to open a hidden row to list the **files** in the snapshot
- link to **checkout** this particular snapshot
- **context** of the snapshot
- **version** of the snapshot
- **revision** of the snapshot
- **login** who checked in the snapshot
- **timestamp** of the checkin
- **description** written by the user at the time of the snapshot
- indicator whether the snapshot is the **current** version for that context

- toggle to open the notes using the **Note Sheet** Widget

## Implementation

The Checkin History Widget can be found as a common column that can be added using the Column Manager.

## Options

There are no options provided for the Checkin History Widget.

## Advanced

```
<element name="history" edit="false">
  <display class="HiddenRowToggleWdg">
    <icon>HISTORY</icon>
    <dynamic_class>tactic.ui.widget.SObjectCheckinHistoryWdg</dynamic_class>
  </display>
</element>
```

## Note

### Description

The Notes Widget allows users to write notes for a particular item (sObject). This widget allows team members to exchange comments for a process by writing them in the Notes Widget. The notes are displayed chronologically with latest one appearing at the top. The complete history is displayed by default. It's one of the common columns which can be added in any view for an sType. A similar note entry widget called the Note Sheet Widget, focuses more on the speed of entry rather than the display of the conversation.

### Info

<b>Name</b>	Notes
<b>Class</b>	tactic.ui.widget.DiscussionWdg
<b>Category</b>	Table Element Widget
<b>Supported Interfaces</b>	TableLayoutWdg
<b>TACTIC Version Support</b>	2.5 +
<b>Required database columns</b>	This widget interacts with the built in sthpw/note table

### Usage

To create a new note, select the New Note button.

This will switch the DiscussionWdg into insert mode where notes and context of the notes can be entered.

In most cases, the grouping for the notes is derived through selecting a 'context'. This context is often chosen in relation to the context of a given 'task' or 'snapshot' (Checkin) for the same parent SObject. This then associates all tasks, notes and snapshots under a specific Search Object. This allows users to retrieve historical data for a Search Object through a context. This answers the question "What's the history of this Asset from the design department?"

To navigate the history of the notes, click on a particular note and it will expand and display the full note.



### Note

Depending on the configuration, the grouping (context) items will be grouped and separated by a group label represented as << label >>. In that case, selecting the group label will trigger a warning popup.

To unset a value, you can usually select the empty value with the label '-- Select --'.

### Implementation

The Notes widget is a common column which can be added using the Column Manager. The item name is "notes". A "default" context is used in this simple implementation.

### Options

<b>context</b>	a global context can be specified
<b>append_context</b>	a context can be appended to the current list (deprecated)
<b>setting</b>	A project setting can be used to drive the contexts. This provides the key of the project setting.



<b>append_setting</b>	This serves the same purpose as setting but would append the contexts at the end
<b>include_submission</b>	If set to true, it would include the notes for the submission (a child) of the current subject.

---

## Advanced

```
<element name="discussion" edit="false">
  <display class="pyasm.widget.DiscussionWdg">
    <context>default</context>
  </display>
</element>
```

# Note Sheet Widget

## Description

The Note Sheet Widget allows entering of many notes in different contexts and different parents at the same time. It can be used for entering notes for any search types. By default, it uses the parent's pipeline processes as the contexts for note entry. Notes can be saved either individually or altogether. There is an option to make a note private as well.

## Info

<b>Name</b>	Note Sheet Widget
<b>Common Title</b>	Note Sheet
<b>Class</b>	tactic.ui.app.NoteSheetWdg
<b>Category</b>	Table Element Widget
<b>Supported Interfaces</b>	TableLayoutWdg
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	none

## Usage

When used with regular sTypes with its pipeline\_code set, the Note Sheet Widget automatically displays the pipeline processes as note context options. Each enabled context is marked with a check in the check box, which goes along with a text box for note entry. Indicate which contexts display for input by selecting the appropriate check boxes. When used with a child search\_type like a task, the Note Sheet Widget assumes its context attribute as the note context.

Clicking on "**save**" icon will save all of the notes together for this parent. To save one note at a time, click on the individual **save** button under the corresponding note.

The private check box turns a note access as private if checked. The history button is used to display all the note entries for a context.

## Implementation

The Note Sheet Widget is a common column that can be added using the Column Manager.

## Options





<b>dynamic_class</b>	Set the class name of the widget to be displayed
----------------------	--

<b>pipeline_code</b>	Specifies a particular pipeline_code to use or if the parent of this note sheet widget does not have the 'pipeline_code' attribute e.g. 'model'. If unspecified, it will be based on the pipeline_code value of its parent.
<b>element_class</b>	To override the default element class NoteTableElementWdg, modify the look or add extra buttons to the UI to enter notes. One method is just to override the method get_action_wdg()
<b>use_parent</b>	When a note sheet is added to a sType like task or snapshot but it is set up so that the note is targeted at its parent, which could be an asset or shot. If so, set this display option to true.
<b>append_context</b>	Used to add contexts that are not defined in the pipeline. Separate the contexts with a pipe character if there are more than one, e.g. producer director.

## Advanced

```
<element name="notes_sheet" edit="false">
  <display class="HiddenRowToggleWdg">
    <dynamic_class>tactic.ui.app.NoteSheetWdg</dynamic_class>
  </display>
</element>
```

## Preview

<input type="checkbox"/> Preview	Title
<input type="checkbox"/> 	Nuclear Explosion
<input type="checkbox"/> 	Car Chase
<input type="checkbox"/> 	Car Warehouse
<input type="checkbox"/> 	Sport Car Warehouse

## Description

The Thumbnail Widget is available for most types by default as the preview tool for images which have been uploaded for preview and thumbnail purposes. An icon for the corresponding file type is displayed for non-image files.

## Info

<b>Name</b>	Thumbnail Widget
<b>Common Title</b>	Preview, Snapshot, Files
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	none

## Implementation

The Thumbnail widget is available in the common columns.

## Options

<b>script_path</b>	Specify a script to control what UI it draws or what happens when the user click on the preview icon. Refer to it by this script path.
<b>detail_class_name</b>	Specify the default behavior to open up a pop-up window but just with a different widget written in Python.
<b>icon_context</b>	The context that the widget displays
<b>icon_size</b>	Control the icon size by percentage (up to 100%) e.g. 30%
<b>min_icon_size</b>	Minimum icon size (in pixels).
<b>latest_icon</b>	If set to 'true', the icon displayed corresponds to the latest checkin in the checkin history. It will disregard the icon context designated for this search type.
<b>filename</b>	If set to 'true', the file name of the linked file is displayed under the icon.
<b>original</b>	If set to 'true', the link will point to the original file with the 'main' file type checked in. Otherwise the scaled down 'web' version of the file will be linked. This is only applicable to image-type files where an icon has been generated during a check-in.

<b>file_type</b>	Whether to display the file type for download or not.
<b>detail</b>	If set to 'false', clicking of the thumbnail will link the underlying picture instead of displaying the single asset view in a pop-up
<b>protocol</b>	'http'(default) or 'file'. The protocol under which the thumbnail link will open when being clicked on. When 'file' is set, the default application is usually Windows explorer or at times Internet Explorer. 'file' mode can alleviate the bandwidth usage on the web server when viewing large media files like Quick Time.
<b>redirect_expr</b>	Works similarly as the redirect but in the form of expression. e.g. @SUBJECT(prod/sequence). If this display option is set for the ThumbWdg for prod/shot, it will display the icon of its sequence instead.

## Task Edit

The screenshot displays the 'Task Edit' interface in TACTIC. At the top, there's a summary for a task titled 'Nuclear Explosion'. It includes a video preview, a description 'Stock footage of a nuclear explosion', keywords 'explosion, fire, nuclear, bomb', and a pipeline with three stages: 'design' (Review, 07/29 - 07/29), 'rough' (Ready, 08/01 - 08/02), and 'final' (Waiting, 07/26 - 07/30). Below the summary is a Gantt chart showing the task's progress over time. The chart has columns for days of the week and dates. The Gantt chart shows four tasks: 'design' (Review, 07/29 - 07/29), 'rough' (Ready, 08/01 - 08/02), 'final' (Waiting, 07/26 - 07/30), and 'delivery' (Waiting, 08/04 - 08/04). Each task has a duration bar and a status indicator.

## Description

The Task Edit Widget is a toggle that opens a hidden row that displays all the tasks for an item. If there are multiple processes for an item, the tasks for those processes will be displayed.

## Info

<b>Name</b>	Task Edit
<b>Common Title</b>	Tasks
<b>Class</b>	tactic.ui.panel.TableLayoutWdg
<b>Category</b>	Table Layout Widget
<b>Supported Interfaces</b>	TableLayoutWdg
<b>TACTIC Version Support</b>	3.0.0 +
<b>Required database columns</b>	none

## Usage

The following details are displayed by the Task Edit Widget for a task:

- a link to the task's Work Area (where the Checkin and Checkout tools can be found)
- the task's description
- status for that process
- the user assigned to the process
- the supervisor of that process
- the priority
- start and end date for the process in the form of a Gantt chart

## Implementation

The Task Edit Widget is a common column that can be added using the Column Manager.

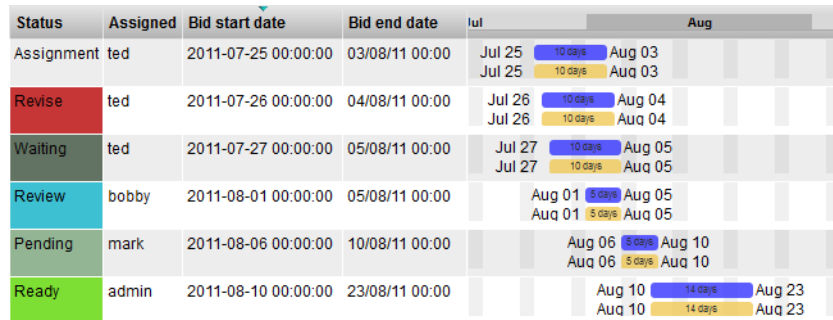
## Options

There are no options provided for the Task Edit Widget.

## Advanced

```
<element name="task_edit" title="Tasks" edit="false">
  <display class="HiddenRowToggleWdg">
    <dynamic_class>tactic.ui.panel.TableLayoutWdg</dynamic_class>
  </display>
</element>
```

## Task Schedule



## Description

The Task Schedule displays a horizontal bar graph representing the schedule of start/end date and duration for all tasks assigned to an item. This widget is a simple pre-configuration of the Gantt Chart widget.

## Info

<b>Name</b>	Task Schedule
<b>Class</b>	tactic.ui.table.GanttElementWdg
<b>Category</b>	Common Columns
<b>TACTIC Version Support</b>	3.0+
<b>Required database columns</b>	none

## Implementation

The Task Schedule Widget is a common column that can be added using the Column Manager.

## Options

The following is the configuration option which makes this widget distinct from its derivative, the Gantt Chart widget.

```
[
  {
    "start_date_expr": "@MIN(sthpw/task.bid_start_date)",
    "end_date_expr": "@MAX(sthpw/task.bid_end_date)",
    "color": "#33F",
    "edit": "true",
    "default": "false"
  },
  {
    "start_date_expr": "@MIN(sthpw/task['context','model'].bid_start_date)",
    "end_date_expr": "@MAX(sthpw/task['context','model'].bid_end_date)",
    "color": "#F0C956",
    "edit": "true",
    "default": "false"
  }
]
```

<b>Show Title</b>	<b>True or False</b> Display the title in the column header.
-------------------	--



<b>Date Mode</b>	<b>visible, hover</b> Always display the start/end date next to the horizontal bar or display the dates only on cursor hover.
<b>Range Start Date</b>	Select the start date range for the tasks to display.
<b>Range End Date</b>	Select the end date range for the tasks to display.
<b>Show Milestones</b>	<b>task, project</b> Display a red vertical bar representing the milestone for the task or the project
<b>Year Display</b>	<b>none, default</b> Display the year in the column header.
<b>Week Display</b>	<b>none, default</b> Display the week in the column header.

## Advanced

```

<element name="task_schedule">
  <display class="tactic.ui.table.GanttElementWdg">
    <options>[
      {
        "start_date_expr": "@MIN(sthpw/task.bid_start_date)",
        "end_date_expr": "@MAX(sthpw/task.bid_end_date)",
        "color": "#33F",
        "edit": "true",
        "default": "false"
      },
      {
        "start_date_expr": "@MIN(sthpw/task['context','model'].bid_start_date)",
        "end_date_expr": "@MAX(sthpw/task['context','model'].bid_end_date)",
        "color": "#F0C956",
        "edit": "true",
        "default": "false"
      }
    ]</options>
  </display>
  <action class="tactic.ui.table.GanttCbK">
    <sObjects>@SOBJECT(sthpw/task)</sObjects>
    <options>[
      {
        "prefix": "bid",
        "sObjects": "@SOBJECT(sthpw/task)",
        "mode": "cascade"
      },
      {
        "prefix": "bid",
        "sObjects": "@SOBJECT(sthpw/task['context','model'])",
        "mode": "cascade"
      }
    ]</options>
  </action>
</element>

```

# Task Status Edit

## Description

The Task Status Edit column is used to display the status of all tasks for the item. It also provides conveniences such as changing the status of the task and the assigned user.

## Info

<b>Name</b>	Task Element Widget
<b>Common Title</b>	Task Status Edit
<b>Class</b>	tactic.ui.table.TaskElementWdg
<b>Category</b>	Common Columns
<b>TACTIC Version Support</b>	3.0.0 +
<b>Required database columns</b>	none

## Usage

Once this column is added into the view, the drop down list can be used to change the status. In addition, this column also displays the process, schedule and the assigned user.

## Implementation

This widget can be added using the Column Manager and can be found under the common columns as **Task Status Edit**.

### Color

TACTIC provides the ability to assign each status its own color. Setting colors is handles from the Project Workflow (Pipeline) editor. Each process in a regular pipeline or a status pipeline can be assigned a color which will be used in this widget.

## Options

<b>Bg Color</b>	<b>status</b> and <b>process</b> . Set what controls the background color of the task. <b>Status</b> sets the task color to be the same as the status color. <b>Process</b> mode sets the task color to be the same color of the process as set in the Workflow Editor.
<b>Status Color</b>	<b>status</b> and <b>process</b> . Set what controls the background color of the status drop down. <b>Status</b> sets the status drop down color to be the same as the status color. <b>Process</b> mode sets the status drop down color to be the same color of the process as set in the Workflow Editor.
<b>Context Color</b>	<b>status</b> and <b>process</b> . Set what controls the background color of the context grid. <b>Status</b> sets the context grid color to be the same as the status color. <b>Process</b> mode sets the context grid color to be the same color of the process as set in the Workflow Editor.
<b>Text Color</b>	Specifies the color of the task text using a color swatch.
<b>Show Process</b>	<b>True</b> or <b>false</b> . Displays the process of the task within the column.
<b>Show Context</b>	<b>True</b> or <b>false</b> . Displays the context of the task within the column.
<b>Show Dates</b>	<b>True</b> or <b>false</b> . Displays the time frame for the task. The schedule will display the start and end date.
<b>Show Assigned</b>	<b>True</b> or <b>false</b> . Displays the assigned user to the task.
<b>Show Track</b>	<b>True</b> or <b>false</b> .. Displays a button on each task which displays the last status and the user who changed it.
<b>Show Labels</b>	<b>True</b> or <b>false</b> . Displays the label of the pipeline's process.
<b>Show Border</b>	<b>all</b> , <b>one-sided</b> , <b>none</b> . <b>All</b> displays a border around each task. <b>One-sided</b> displays a border around one one side of the task. <b>None</b> hides the border.
<b>Show Current Pipeline Only</b>	<b>True</b> or <b>false</b> . Displays tasks for the current pipeline only.
<b>Show Task Edit</b>	<b>True</b> or <b>false</b> . Displays a button which pops-up a window to edit the task info.
<b>Task Edit view</b>	Specify the Task view by which to edit the task information.
<b>Task Filter</b>	<b>panel</b> , <b>vertical</b> , <b>horizontal</b> : Layout orientation to display the list of tasks.
<b>Layout</b>	<b>context only</b> or <b>process only</b> : Displays only tasks for either the context or the process.
<b>Edit Status</b>	<b>True</b> or <b>false</b> . Allows the user to open the status drop down selection box for the status to change it.
<b>Edit Assigned</b>	<b>True</b> or <b>false</b> . Allows the user to open the status drop down selection box for the assigned user to change it.

## Advanced

```
<element name='task_status_edit'>
  <display class='tactic.ui.table.TaskElementWdg'>
    <show_context>true</show_context>
    <show_assigned>true</show_assigned>
    <show_dates>true</show_dates>
    <edit>true</edit>
  </display>
  <action class='tactic.ui.table.TaskElementCbK' />
</element>
```

## Task Status History

12 items found

Showing: 

1 - 12

<input type="checkbox"/>	Preview	Category	Asset Code	Title	Checkin	Tasks	Task Pipeline				Task status history
<input type="checkbox"/>		video	asset001	Nuclear Explosion			<div>design 07/29 - 07/29 <div>Review</div> beth</div>	<div>rough 08/01 - 08/02 <div>Ready</div> mark</div>	<div>final 07/26 - 07/30 <div>Waiting</div> beth</div>	<div>delivery 08/04 - 08/04 <div>Waiting</div> bobby</div>	
<input type="checkbox"/>	Search type	Search id	Statu	Changed By	Timestamp	To status	From status	Project code			
<input type="checkbox"/>	sthpw/task	1042		admin	Dec 07, 2011 - 23:40	Waiting	Review	project			
<input type="checkbox"/>	sthpw/task	1042		admin	Aug 30, 2011 - 22:55	Review	Approved	project			
<input type="checkbox"/>	sthpw/task	1042		admin	Aug 30, 2011 - 22:31	Approved	Review	project			
<input type="checkbox"/>	sthpw/task	1042		admin	Aug 08, 2011 - 17:46	Review	Approved	project			
<input type="checkbox"/>	sthpw/task	1042		mark	Aug 07, 2011 - 00:00	Pending	Waiting	project			

## Description

The Task Status History is a toggle that opens a hidden row that displays all the status changes for an item. If there are multiple processes for an item, the status updates for those processes will be displayed.

## Info

<b>Name</b>	Task Status History
<b>Class</b>	tactic.ui.panel.TableLayoutWdg
<b>Category</b>	Common Columns
<b>TACTIC Version Support</b>	3.0+
<b>Required database columns</b>	none

## Implementation

The Task Edit Widget is a common column that can be added using the Column Manager.

## Options

There are no options provided for the Task Edit Widget.

## Advanced

```
<element name="task_status_history">
  <display class="HiddenRowToggleWdg">
    <dynamic_class>tactic.ui.panel.TableLayoutWdg</dynamic_class>
    <search_type>sthpw/status_log</search_type>
    <view>table</view>
    <expression>@SOBJECT(sthpw/task.sthpw/status_log)</expression>
    <mode>simple</mode>
  </display>
</element>
```

## Work Button

### Description

The Work Element Widget is used for accessing the checkin and checkout tool needed to handle a task assigned. After a task is assigned, an artist can go to the "My Tasks" or any other task page where there is a "Work" column which will expand to this widget. You can carry out several typical functions related to check-in and check-out in the sub tabs that open. You can even customize what tabs are opened when the work button is clicked on.

### Info

<b>Name</b>	Work
<b>Class</b>	tactic.ui.table.WorkElementWdg
<b>Category</b>	Table Element Widget
<b>Supported Interfaces</b>	TableLayoutWdg
<b>TACTIC Version Support</b>	3.5.0 +
<b>Required database columns</b>	none

### Usage

When clicked on, it opens up a new Work area tab with 3 sub tabs underneath which comprise all the functions an artist would need when assigned a task. He can enter notes, change task status, review check-in history, check in, and check out files.

The General Check-in Widget appears in the Check-in sub tab. You can click "Browse" here to select the file to be checked in. The `is_current` checkbox in Options can be used to make a snapshot current on checking in. The link checkbox, when checked, links the sandbox directory to the process tied to the task. It makes it easy for an artist to jump to a different process and checks out their snapshots into the current sandbox associated with the task. Otherwise, if you check out a file from the model process, it will be copied to the model sandbox folder.

### Options

<b>checkout_panel_script_path</b>	Deprecated in 3.5. Use the <code>tab_config_&lt;&gt;</code> method to set up custom checkout tab
<b>checkout_script_path</b>	A custom check-out script path you can specify to override the default check-out script. The default check-out script checks out everything under the selected snapshot. Refer to Example 4.
<b>validate_script_path</b>	A script path pointing to a JS script that is run before the actual check-in. If it throws an error using <code>"throw(&lt;error message&gt;)"</code> , the check-in will not initiate. You can also use it to run some client-side preprocessing of the file or directory.
<b>transfer_mode</b>	Upload, copy, move, preallocate are supported. 'preallocate' can only be used if the client machine has direct disk write access to the TACTIC asset repo. It skips the need to hand off the files in the handoff directory. 'copy' is recommended for most situation when users are usually granted only read access to the TACTIC asset repo.
<b>mode</b>	Sequence, file, dir, and add are supported. 'sequence' is for file sequence checkin; 'file' is for single file checkin, 'dir' is for directory checkin and 'add' if for appending file or dir to an existing snapshot. If not specified, multiple selections will be available for the user to choose. Note: upload transfer mode only supports single file or file sequence check-in.

<b>checkin_panel_script_path</b>	Deprecated in 3.5. Use the <code>tab_config_&lt;&gt;</code> method to set up custom checkin tab
<b>checkin_script_path</b>	A custom checkin script path you can specify to override the default check-in script. This can be used in conjunction with the <code>validate_script_path</code> .
<b>process</b>	If set, the process specified will be pre-selected when the General Checkin Widget is drawn,
<b>lock_process</b>	If set to true, the user will not be able to choose a different process during check-in on the General Checkin-in Widget
<b>checkin_relative_dir</b>	If specified, e.g. WIP, it is appended to the current sandbox directory and preselected as the directory to be checked in. It's applicable to Direcotry-type checkin
<b>checkin_ui_options</b>	It applies to the check-in options of the CheckinWdg. Supported attribute at the moment is "is_current" e.g. {"is_current":"false"} would make all check-ins non-current. {"is_current":"optional"} would make the checkbox unchecked by default. Not specifying it would render the option available to the user to choose at the check-in time.
<b>show_versionless_folder</b>	If set to true, it displays the latest and current versionless folders.

## Implementation

The following defines the default "Work" element. It looks a bit complicated but in most cases, you would just need to simply change the different options available through "Edit Column Definition":

```
<element name="work" title="Work on Task">
  <display class="tactic.ui.table.WorkElementWdg">
    <transfer_mode>upload</transfer_mode>
    <cbjs_action>
      var tbody = bvr.src_el.getParent(".spt_table_tbody");
      var element_name = tbody.getAttribute("spt_element_name");
      var search_key = tbody.getAttribute("spt_search_key");
      var checkin_script_path = bvr.checkin_script_path;
      var checkin_ui_options = bvr.checkin_ui_options;
      var validate_script_path = bvr.validate_script_path;
      var checkout_script_path = bvr.checkout_script_path;
      var checkin_mode = bvr.mode;
      var transfer_mode = bvr.transfer_mode;
      var sandbox_dir = bvr.sandbox_dir;
      var lock_process = bvr.lock_process;

      var server = TacticServerStub.get();
      var code = server.eval( "@GET(parent.code)", {search_keys: search_key} );

      spt.tab.set_main_body_tab();
      spt.tab.add_new();
      var kwargs = {
        'search_key': search_key,
        'checkin_script_path': checkin_script_path ,
        'checkin_ui_options': checkin_ui_options ,
        'validate_script_path': validate_script_path ,
        'checkout_script_path': checkout_script_path,
        'mode': checkin_mode ,
        'transfer_mode': transfer_mode,
        'sandbox_dir': sandbox_dir,
        'lock_process': lock_process

      }

      var title = "Task: " + code;
      var class_name = "tactic.ui.tools.sobject_wdg.TaskDetailWdg";
      spt.tab.load_selected(search_key, title, class_name, kwargs);
    </cbjs_action>
  </display>
</element>
```

```
<icon>WORK</icon>
</display>
</element>
```

The following defines a different usage of it using copy transfer mode, a custom checkout script and a custom validating checkin script. The value of the two script paths are the script\_path you have saved in the Script Editor. lock\_process is set to false. To enable these options, you can do it in the context menu "Edit Column Definition" and set the following:

```
checkout_script_path: checkout/all_processes
validate_script_path: checkin/validate_frames
transfer_mode: copy
lock_process: false
```

The following shows a way to customize what the small check-out button does in the checkout\_tool view. In widget config, we will set the column definition for the element checkout for the "sthpw/snapshot" search type. It can be accessed through "Edit Column Definition".

```
checkout_script_path: checkout/checkout_tool_script
```

## Script Samples

### Example 1: checkin/validate\_frames

```
var values = bvr.values;
var file_path = values.file_paths[0];
var sk = values.search_key;
var applet = spt.Applet.get();

var file_list = applet.list_dir(file_path);
var server = TacticServerStub.get();
var st = 'prod/shot';
var shot = server.get_by_search_key(sk);
var frame_count = parseInt(shot.frame_count, 10);
for (var i=0; i <file_list.length; i++){

    var base =spt.path.get_basename(file_list[i]);
    if ( base == 'FRAMES') {
        var frames = applet.list_dir(file_list[i]);

        if (frames.length != frame_count) {
            throw('Frames length in FRAMES [' + frames.length
                + '] folder does not match shot\'s frame count');
        }
    }
}
```



Example 2: checkout/all\_processes. It illustrates how to implement a custom check-out that only checks out a portion of what has been checked in.

```
//back up the Work-in-progress folder
function backup_WIP(bvr) {
    var sandbox_dir = bvr.sandbox_dir;
    var applet = spt.Applet.get();
    var found_WIP = false;
    var dirs = applet.list_dir(sandbox_dir, 0);

    for (var k=0; k < dirs.length; k++){
        if (/WIP$/i.test(dirs[k])){
            found_WIP = true;
            break;
        }
    }
    if (!found_WIP) {
        alert('WIP folder not found. Backing up of WIP folder aborted')
    }
    else {

        var server = TacticServerStub.get();
        var folder = spt.path.get_basename(sandbox_dir);

        var date_obj = new Date();
        var suffix = date_obj.getFullYear().toString()
            + spt.zero_pad((date_obj.getMonth() + 1).toString(), 2)
            + spt.zero_pad(date_obj.getDate().toString(), 2) + '_' +
            spt.zero_pad(date_obj.getHours().toString(), 2)
            + spt.zero_pad(date_obj.getMinutes().toString(), 2);

        var parts = sandbox_dir.split(/[\\\/]/);

        sandbox_dir = sandbox_dir + '/WIP';
        var backup_dir = parts.join('/') + '/WIP' + '_' + suffix;

        applet.copytree(sandbox_dir, backup_dir);

        //remove the contents of WIP
        applet.rmtree(sandbox_dir);
        applet.makedirs(sandbox_dir);
    }
}

// just checkout a subfolder named REF. if it's not found, just check out the
// first subfolder
function checkout_snapshot_table(bvr){

    var top = bvr.src_el.getParent(".spt_checkin_top");
    var table = top.getElement(".spt_table");
    var search_keys = spt.dg_table.get_selected_search_keys(table);
    if (search_keys.length == 0) {
        alert('Please check the checkbox(es) to check out a version.');
```

```

var sandbox_input = top.getElement('.spt_sandbox_dir');
if (sandbox_input)
    bvr.sandbox_dir = sandbox_input.value;
for (var i = 0; i < search_keys.length; i++) {

    checkout_snapshot(bvr, search_keys[i]);
}
spt.app_busy.hide();
}

function checkout_snapshot(bvr, snapshot_key, downlevel) {
    var server = TacticServerStub.get();

    try {
        var paths = server.get_all_paths_from_snapshot(snapshot_key);

        //var sandbox_dir =
server.get_client_dir(snapshot_key, {mode: 'sandbox'});
        var sandbox_dir = bvr.sandbox_dir;
        var applet = spt.Applet.get();

        for (var i = 0; i < paths.length; i++) {
            var path = paths[i];
            var parts = path.split(/[\\\/]/);
            var dirs = applet.list_dir(path);

            var tar_dir = '';
            for (var j=0; j < dirs.length; j++) {
                if ((/REF/i).test(dirs[j]))
                    tar_dir = dirs[j];
            }
            //just take the first one if REF is not found
            if (!tar_dir) {
                alert('REF not found. First subfolder is checked out');
                tar_dir = dirs[0];
            }

            var folder = spt.path.get_basename(tar_dir);
            var new_path = path + '/' + folder;

            var sand_paths = applet.list_dir(sandbox_dir, 0);
            for (var j=0; j< sand_paths.length; j++) {
                var dst_folder = spt.path.get_basename(sand_paths[j]);
                if (dst_folder == 'REF') {
                    alert('REF folder already exists in ['
                        + sandbox_dir + '] Please rename or remove it first.');
```

return;

```

                }
            }

            // the applet can decide between copy_file or copytree
            applet.copytree(new_path, sandbox_dir + "/" + folder);

        }
    }
    catch(e){
        alert(spt.exception.handler(e));
    }
}

backup_WIP(bvr);
var down_level = 1;

```

```
checkout_snapshot_table(bvr, down_level);
```

Example 3: Custom Checkout button callback passing a specific script for the Check-out Widget popup using display option "checkout\_panel\_script\_path"

```
var class_name = 'tactic.ui.widget.CheckoutWdg';

var values = bvr.values;

var search_key = values.search_key;
var sandbox_dir = values.sandbox_dir;
var process = values.process;

var options = { 'show_publish': 'false',
  'process': process,
  'search_key': search_key,
  'checkout_script_path': 'checkout/custom_checkout',
  'sandbox_dir': sandbox_dir
};

var popup_id = 'Check-out Widget';
spt.panel.load_popup(popup_id, class_name, options);
```

Example 4: Custom check-out script for the small check-out button in the checkout\_tool view. This can be used to customize a quick-checkout for the latest or current snapshot without opening the Check-out popup widget, using display option "checkout\_script\_path"

```
function checkout_snapshot(bvr) {
    var values = bvr.values;

    var snapshot_key = values.search_key;
    var context = values.context;

    var server = TacticServerStub.get();
    // get the files for this snapshot, always get the latest
    // instead of relying on the last snapshot when the UI was drawn

    try {
        var paths = server.get_all_paths_from_snapshot(snapshot_key);

        //var sandbox_dir = server.get_client_dir(snapshot_key, {mode:'sandbox'});
        // This one comes from values as the sandbox_dir is determined by
        // the snapshot only
        var sandbox_dir = values.sandbox_dir;

        var applet = spt.Applet.get();

        for (var i = 0; i < paths.length; i++) {
            var path = paths[i];
            var parts = path.split(/[\\\/]/);
            var dirs = applet.list_dir(path);

            var tar_dir = '';
            for (var j=0; j < dirs.length; j++) {
                if ((/REF/i).test(dirs[j]))
                    tar_dir = dirs[j];
            }
            //just take the first one if REF is not found
            if (!tar_dir) {
                alert('REF not found. First subfolder is checked out');
                tar_dir = dirs[0];
            }
            var folder = spt.path.get_basename(tar_dir)
            var new_path = path + '/' + folder;

            var sand_paths = applet.list_dir(sandbox_dir, 0);
            for (var j=0; j< sand_paths.length; j++) {
                var dst_folder = spt.path.get_basename(sand_paths[j]);
                if (dst_folder == 'REF') {
                    alert('REF folder already exists in [' + sandbox_dir + '] Please rename or
remove it first to avoid mixing files.');
```

Example 5: Custom checkin\_script using display option "checkin\_script\_path". The default snapshot\_type is file, if the file extension is .mov, the snapshot\_type is set to 'mov'.

```
var file_paths = bvr.values.file_paths;
var description = bvr.values.description;
var search_key = bvr.values.search_key;
var context = bvr.values.context;
var transfer_mode = bvr.values.transfer_mode
var is_current = bvr.values.is_current;
var path = file_paths[0]
spt.app_busy.show("File Checkin", path);

var snapshot_type = 'file';
if (path.test(/\\.mov$/)){
    snapshot_type = 'mov';
}
var server = TacticServerStub.get();
snapshot = server.simple_checkin(search_key, context, path,
{description: description, mode: transfer_mode, is_current: is_current,
 snapshot_type: 'mov'});
```

## Work Hours List

Priority	Status	Due Date	Feb	Sun 26	Mon 27	Tue 28	Wed 29	Thu 01	Fri 02	Sat 03	Parent Id
4	ready	2011-08-29									tactic-set
4	in_progress	2012-01-08				4.0	3.0	4.0		11.0	tactic-set
0	In Progress	2012-01-16		1.0						1.0	Support

## Description

The Work Hours widget provides an interface to record the number of work hours spent for each task. The breakdown of the work hours by task allows the analysis to be broken down at the lowest level of detail.

## Info

<b>Name</b>	Work Hours List
<b>Class</b>	tactic.ui.table.WorkHoursElementWdg
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	none

## Implementation

The Work Hours List Element is a common column that can be added to any task view using the Column Manager.

## Options

There are no options available for this widget.

<b>view</b>	The view to retrieve from the Widget Config. This is not required if the HTML option is supplied.
<b>html</b>	This option is where the HTML code is embedded.
<b>search_type</b>	The Search Type the CustomLayoutWdg applies to (if applicable)

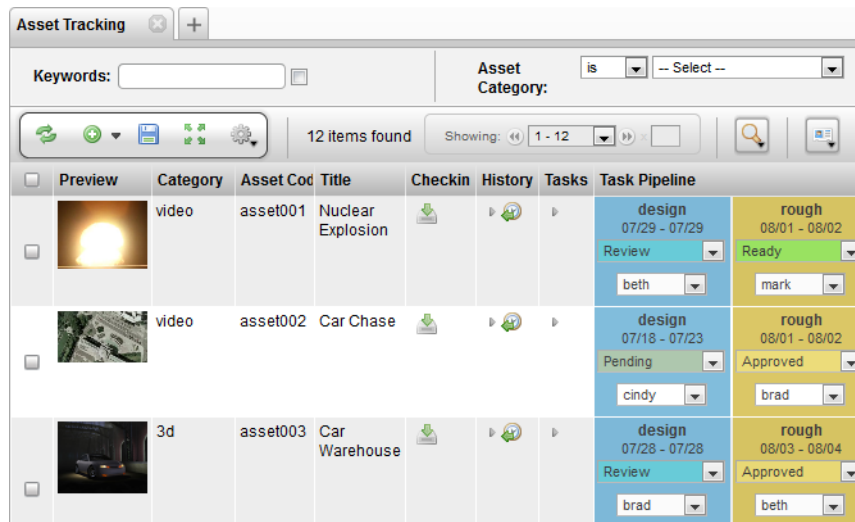
## Examples

We can record 4 hours of work on Wednesday and 3 hours on Thursday for a task. The total for that week will also be displayed as a convenience.

Feb	Sun 12	Mon 13	Tue 14	Wed 15	Thu 16	Fri 17	Sat 18
			4.0	3.0			7.0

# Layout Widgets

## View Panel



## Description

The View Panel is a composite widget which binds together a Table Layout Widget and a Search Widget. The Search Widget is a searching mechanism that retrieves items and transfers them to a Table Layout Widget to draw. The View Panel Widget is used in most of TACTIC's predefined views.

## Info

Name	View Panel
Class	ViewPanelWdg
TACTIC Version Support	2.5.0 +
Required database columns	none

## Implementation

The View Panel widget makes use of the TableLayoutWdg capabilities. The views available to the View Panel are identical to that of the Table Layout Widget.

## Options

show_gear	Flag to show the gear menu.
show_search	Flag to show the search box.
show_search_limit	Flag to show the search limit.
show_insert	Flag to show the insert button.
insert_view	Specify the path to a custom insert view.
edit_view	Specify the path to a custom edit view.
show_commit_all	Flag to show the commit all button.

<b>show_refresh</b>	Display the refresh button on the shelf.
<b>show_row_select</b>	Flag to show row_selection.
<b>popup</b>	Pop the view up in a pop-up window.
<b>layout</b>	<b>default, tile, static, raw, fast_table, old_table</b>
<b>search_type</b>	The type that this widget works with
<b>view</b>	The TACTIC name for the view. e.g. admin.test_asset_tracking
<b>do_initial_search</b>	Run the search on loading of the view.
<b>simple_search_view</b>	Specify the simple search view.
<b>custom_filter_view</b>	View for custom filters. Defaults to "custom_filter".
<b>process</b>	The process which is applicable in the UI when load view is used.
<b>mode</b>	<b>simple, insert</b>
<b>parent_key</b>	Provides a parent item to filter in the search.
<b>search_key</b>	Provides the starting search key.
<b>element_names</b>	Provides a list of column names (ie. "preview,name,description") for the view.
<b>schema_default_view</b>	(INTERNAL) flag to show whether this is generated straight from the schema.
<b>order_by</b>	The column name to order ascending by.
<b>search_view</b>	(INTERNAL) View for custom searches.
<b>width</b>	Set the default width of the table
<b>expression</b>	Use an expression for the search. The expression must return items.
<b>filter</b>	JSON data structure representing the settings for SearchWdg

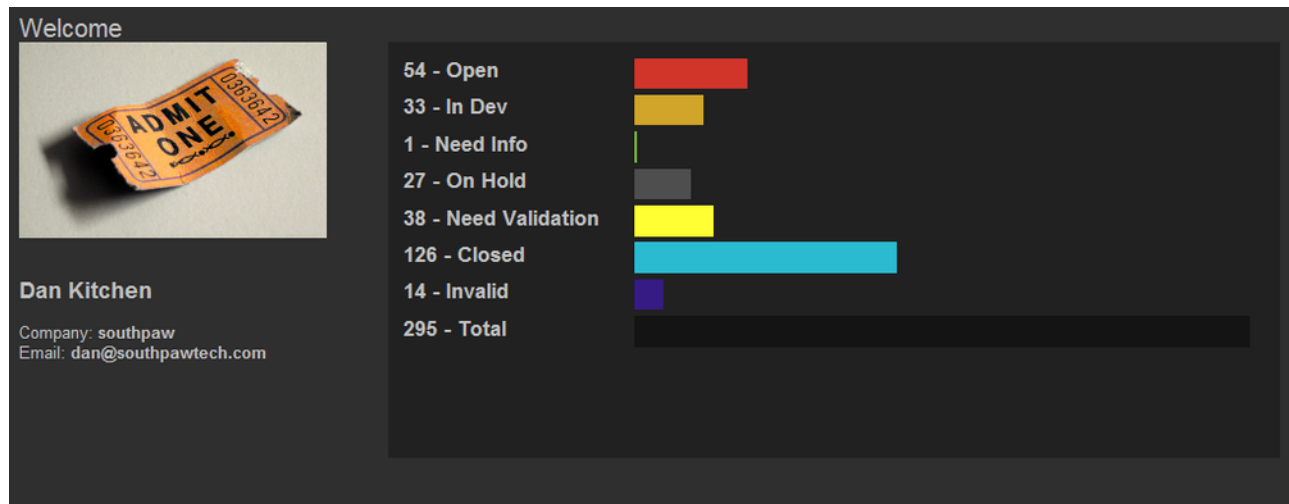
## Advanced

Often, the ViewPanelWdg is defined from a side bar link. It can be defined by XML as follows

```
<element name='summary'>
  <display class='tactic.ui.panel.ViewPanelWdg'>
    <search_type>sthpw/task</search_type>
    <view>task_summary</view>
  </display>
</element>
```



## Custom Layout



## Description

The Custom Layout Widget is a simple tool which opens up an incredible amount of customizable interface flexibility and integration from within the TACTIC UI. This widget provides a container in the web page which supports embedding of HTML code including TACTIC Widgets, Expressions and Behaviours. This allows development of complex widgets similar to the standard widgets delivered with TACTIC.

With this, the following examples can be achieved:

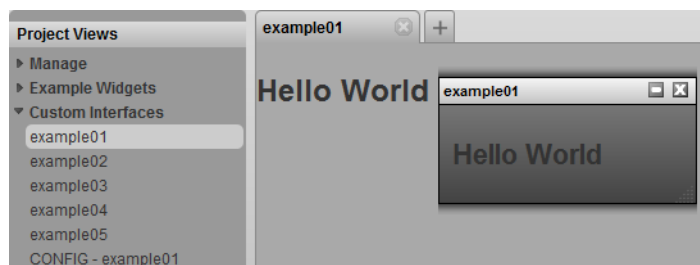
- Custom reports can be made which include dynamic TACTIC Expressions presented in a customized web page design.
- Views can be assembled using a combination of TACTIC widgets along with regular HTML elements.
- Embeddable web code such as Google maps, timezone clocks, web mail clients etc. can be embedded in the TACTIC interface.

## Info

<b>Name</b>	CustomLayoutWdg
<b>Class</b>	tactic.ui.panel.CustomLayoutWdg
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	none

## Implementation

The Custom Layout Widget in its simplest form is a delivery mechanism for HTML code. The following "Hello World" example below demonstrates this.



```
<element name="hello_world">
  <display class="tactic.ui.panel.CustomLayoutWdg">
    <html>
      <h2>Hello World</h2>
    </html>
  </display>
</element>
```

Where a large part of the considerations for usage of a Custom Layout is where it will be embedded and how it will get retrieved. There are a few ways to implement this widget:

- In a cell (element) in a TableLayoutWdg (Example 1a).
- In the widget config, then called from a link in the sidebar (Example 1b).
- In the widget config, then called from a Custom Script (Javascript) (Example 1c).

Once a delivery method has been decided, it also needs to be decided if the custom Layout will need to evaluate based on an item being passed in as the parent or 'starting point' (technically speaking a 'relative' expression). For example, to use a CustomLayout to display a report for a item, the dynamic data in the CustomLayout needs to be derived starting from the specified item.

Taking advantage of relative expressions is usually accessed/assumed through the search\_key value for the widget. This is most often done through TACTIC Expressions embedded in the HTML code or, through JavaScript code interacting with the CustomLayout. Either way, the search\_key value must be passed into the widget for relative behavior. A simple example is that a button in a table can pass in the search key based on the item (row) it was clicked for. This would allow for loading of a custom dashboard which shows all information pertaining to that item.

### Embedded Expressions

Expressions can be embedded in the Custom Layout through usage of a [expr] style tag. This Tag allows for embedding of expressions that are evaluated before the HTML which provide the resulting values into the HTML code.

The following example displays the task status of the modelling process.

```
<element name="hello_world">
  <display class="tactic.ui.panel.CustomLayoutWdg">
    <html>
      <div>
        Model Status: [expr]@GET(sthpw/task['process', 'model'].status)[/expr]
      </div>
    </html>
  </display>
</element>
```

### Embedded Widgets

The CustomLayoutWdg provides full support for embedding of TACTIC Widgets. For example TableLayoutWdg and EditWdg can be placed in a CustomLayout. This allows, for example, the ability to create a 'dashboard' which can show multiple Tables, CustomLayouts HTML, etc.

```

<element name="hello_widget">
  <display class="tactic.ui.table.CustomLayoutWdg">
    <html>
      <element name="tasks">
        <display class="TableLayoutWdg">
          <search_type>sthpw/task</search_type>
          <view>task_list</view>
          <mode>simple</mode>
          <do_search>true</do_search>
          <search_key>{@GET(state.search_key)}</search_key>
        </display>
      </element>
    </html>
  </display>
</element>

```



## Note

In the example above, the `<search_key>` option is automatically being passed the `search_key` from the state of the overall Custom Layout. What this will do is pass in the search key to the table which will automatically filter it to only show items related to the `search_key` (parent). For example in a dashboard for a shot, the tasks table will only display tasks related to the shot as opposed to all tasks in the system.

## Embedded Behaviours

The Custom Layout also supports usage of the TACTIC JavaScript Behaviour system. With this, elements in the Custom Layout can contain embedded behaviours which allow for creation of custom interfaces and utilities. This opens up full a connection with the interface, clientAPI and Java Applet.

```

<?xml version='1.0' encoding='UTF-8'?>
<config>
  <hello_world>
    <html>
      <span>This is a button:</span>
      <input type='button' class='spt_button1' value='Press Me' />
    </html>
    <behavior class='spt_button1'>{
      "type": "click_up",
      "cbjs_action": ''
      alert('Hello World');
    }
  </behavior>
</hello_world>
</config>

```

## Options

<b>view</b>	The view to retrieve from the Widget Config. This is not required if the HTML option is supplied.
<b>html</b>	This option is where the HTML code is embedded.
<b>search_type</b>	The Search Type the CustomLayoutWdg applies to (if applicable)

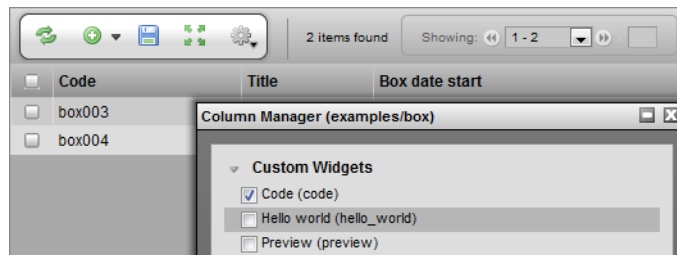
## Examples

### Example 1a

This can be stored in the definition view and called as an element by name (`<element name="hello_world"/>`) or, directly in the view config.

<input type="checkbox"/>	Category	Search type	View	Config	Login
<input type="checkbox"/>		examples/box	definition	<pre> &lt;config&gt;   &lt;definition&gt;     &lt;element name="preview"/&gt;     &lt;element name="code"/&gt;      &lt;element name="hello_world"&gt;       &lt;display class="tactic.ui.panel.CustomLayoutWdg"&gt;         &lt;html&gt;           &lt;h2&gt;Hello World&lt;/h2&gt;         &lt;/html&gt;       &lt;/display&gt;     &lt;/element&gt;   &lt;/definition&gt; &lt;/config&gt; </pre>	admin

If added to the definition, it will be available as a Widget Column in the **Column Manager**.



```

<config>
  <definition>
    <element name="preview"/>
    <element name="code"/>

    <element name="hello_world">
      <display class="tactic.ui.panel.CustomLayoutWdg">
        <html>
          <h2>Hello World</h2>
        </html>
      </display>
    </element>
  </definition>
</config>

```

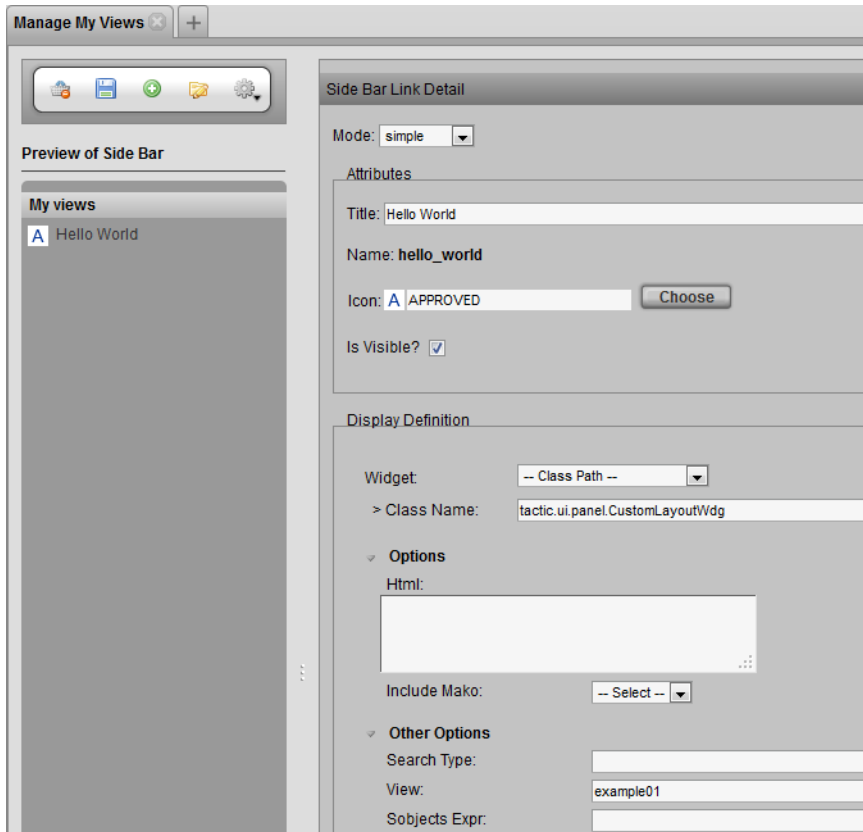
## Example 1b

The following example shows how to create a sidebar link which loads a Custom Layout view defined in the Widget Config.

1. In the Widget Config enter the following hello\_world view. This can be called from The Javascript code and sent to a popup.

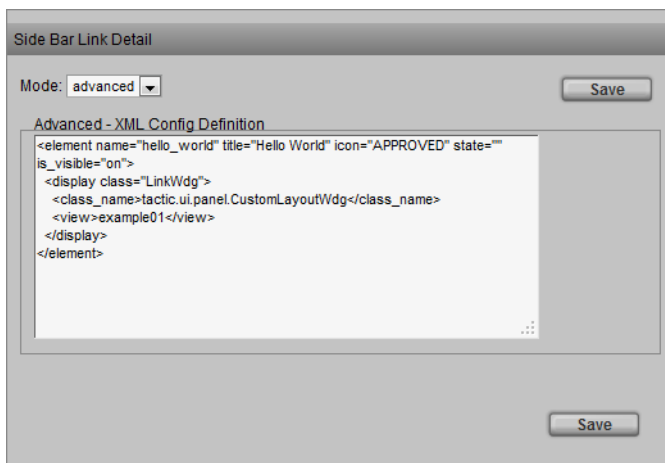
<input type="checkbox"/>	Category	Search type	View	Config	Login
<input type="checkbox"/>		CustomLayoutWdg	example01	<pre> &lt;config&gt;   &lt;example01&gt;     &lt;html&gt;       &lt;h1&gt;Hello World&lt;/h1&gt;     &lt;/html&gt;   &lt;/example01&gt; &lt;/config&gt; </pre>	

2. The link can be configured in the Project Views Manager. Under the action menu, select "New Link". In the popup, Fill the options as shown in the following image.



3. Once the link is saved, select it in the Preview of Side Bar to load its options into the Element Detail panel on the right. Once loaded switch the mode to 'Advanced' and double check that the XML config contains the following:

```
<element name="hello_world" title="Hello World" icon="APPROVED" state="" is_visible="on">
  <display class="LinkWdg">
    <class_name>tactic.ui.panel.CustomLayoutWdg</class_name>
    <view>example01</view>
  </display>
</element>
```



## Example 1c

The following example shows how to create a CustomLayoutWdg View in the widget config then, call it from the Javascript Editor

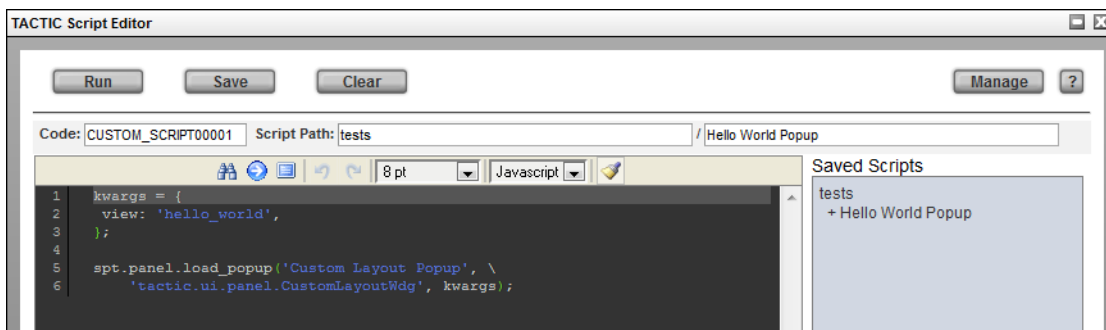
1. In the Widget Config enter the following hello\_world view. This can be called from The Javascript code and sent to a pop-up.

<input type="checkbox"/>	Category	Search type	View	Config	Login
<input type="checkbox"/>		CustomLayoutWdg	hello_world	<pre>&lt;config&gt;   &lt;hello_world&gt;     &lt;html&gt;       &lt;h2&gt;Hello World&lt;/h2&gt;     &lt;/html&gt;   &lt;/hello_world&gt; &lt;/config&gt;</pre>	













2. In the Javascript Editor create the following custom script example to load the view into a pop-up.

```
kwargs = {
  view: 'hello_world',
};

spt.panel.load_popup('Custom Layout Popup', \
  'tactic.ui.panel.CustomLayoutWdg', kwargs);
```



## Table Layout

<input type="checkbox"/>	Preview	Category	Asset Code	Title	Description	Keywords	Checkin	History	Tasks	Task Pipeline
<input type="checkbox"/>		video	asset001	Nuclear Explosion	Stock footage of a nuclear explosion	explosion, fire, nuclear, bomb				<div>design 07/29 - 07/29</div> <div>Review</div> <div>beth</div> <div>bobby</div>
<input type="checkbox"/>		video	asset002	Car Chase	Stock footage of a overhead car chase.	car, chase, fast, race, map				<div>design 07/18 - 07/23</div> <div>Pending</div> <div>cindy</div>
<input type="checkbox"/>		3d	asset003	Car Warehouse	3D Car asset in a warehouse setting	car, warehouse, dark, headlights, exterior, engine				<div>design 07/28 - 07/28</div> <div>Review</div> <div>brad</div> <div>mark</div>

## Description

The TableLayoutWdg is the primary widget used to layout tabular data. It is primarily driven by the widget configuration. The TableLayoutWdg has the ability to display complex widgets inside each cell, to inline edit the data and to color code cells. It is the widget that is most often used to display information within the TACTIC.

## Info

<b>Name</b>	Table Layout
<b>Class</b>	tactic.ui.panel.TableLayoutWdg
<b>TACTIC Version Support</b>	2.5.0 +
<b>Required database columns</b>	none

## Implementation

The TableLayoutWdg makes use of "views" which are defined in the widget config for each project. When the Table is loaded as part of an interface, a view configuration is passed into it which defines which columns and widgets should be displayed in the view. Typically, these view configurations are automatically saved in the background when a user saves a view from within the TACTIC interface. The table itself provides the ability to add, remove, rearrange, resize and group columns which can then be saved out often as links in the sidebar.

The following shows a simplified version for an "asset tracking" view as saved in the background widget config.

```
<config>
  <asset_tracking layout="TableLayoutWdg" >
    <element name="preview" width="74px"/>
    <element name="asset_category_code" width="64px"/>
    <element name="code" width="61px"/>
    <element name="title" width="121.883px"/>
    <element name="description" width="276.75px"/>
    <element name="keywords" width="253.367px"/>
    <element name="general_checkin" width="27px"/>
    <element name="history" width="42px"/>
    <element name="task_edit" width="29px"/>
    <element name="task_status_edit" width="223.167px"/>
  </asset_tracking>
```

```
</config>
```

The widget configuration is an XML document. In this example, it defines an "asset\_tracking" view with elements (preview, asset\_category, code, title, description, keywords, etc...).

To draw what to display, TableLayoutWdg looks at the list of elements defined in the widget config and draws a column for each element. TACTIC then draws a row for each item that was either retrieved from a search, an expression or by supplied items. Each cell in the table represents an item being drawn by the defined element for a given column.

While the top widget configuration defines the list of elements to draw the columns, the exact definition of each element do not necessarily appear here. There are a number of views which define an element. Some of these elements may be defined inline or they may be defined elsewhere. There is a set hierarchy which the TableLayoutWdg looks for to find the definition of a particular element.

The hierarchy which TableLayoutWdg looks to find the definition for an element is as follows:

1. the given type, view combination in the widget\_config table
2. the "definition" view for the given type in the widget\_config table
3. the predefined views for a given type (modules shipped with TACTIC will have predefined views for many of the items to ensure proper functioning of TACTIC even if there are no entries in the widget\_config database)
4. the "default\_definition" for a given type as defined in the predefined views.

The third and fourth locations only apply to predefined types that are shipped with TACTIC. All custom types will only use the first two.

## Options

<b>search_type</b>	Defines the type that this table will be displaying. It is used both for finding the appropriate widget config and for handling search (if necessary). Defaults to "table".
<b>view</b>	Defines the view that this table will displaying. It used to find the appropriate widget config to display the table.
<b>do_search</b>	By default, the TableLayoutWdg will handle the search itself. However, certain widgets may wish to turn this functionality off because they are supplying the search (internally used by ViewPanelWdg)
<b>order_by</b>	Add an explicit order by in the search
<b>expression</b>	Use an expression to drive the search. The expression must return items.
<b>parent_key</b>	Set a specific parent for the search
<b>width</b>	Define an initial overall width for the table
<b>show_row_select</b>	Flag to determine whether or not to show row_selection
<b>show_gear</b>	Flag to determine whether or not to show the gear menu.
<b>show_insert</b>	Flag to determine whether or not to show the insert button.
<b>insert_mode</b>	aux inline pop-up none - set the insert mode of the table
<b>search_limit</b>	An overriding search limit. A value < 0 means no limit affecting the search
<b>config_xml</b>	Explicitly define the widget config
<b>element_names</b>	Explicitly set the element names to be drawn



## Advanced

Very often, the `TableLayoutWdg` is not used directly, but is used through the `ViewPanelWdg`, which combines the `TableLayoutWdg` with the `SearchWdg`. Using `ViewPanelWdg` will provide all the functionality in a table view

Using the `TableLayoutWdg` does provide a simpler view if the search is already known,

This simple example shows the login table and the objects are explicitly given.

```
from tactic.ui.panel import TableLayoutWdg
div = DivWdg()
table = TableLayoutWdg(search_type='sthpw/login', view='table')
sObjects = Search("sthpw/login").get_sObject()
table.set_sObjects(sObjects)
div.add(table)
```

An expression can be set for the search as well.

```
from tactic.ui.panel import TableLayoutWdg
div = DivWdg()
expression = "@SOBJECT(sthpw/login)"
table = TableLayoutWdg(search_type='sthpw/login', view='table', expression=expression)
div.add(table)
```

This example embeds the login table with a "table" view in a `CustomLayoutWdg`.

```
<config>
<login>
  <html>
    <h1>This is the login table</h1>
    <element name='login_table' />
  </html>
  <element name='login_table'>
    <display class='tactic.ui.panel.TableLayoutWdg'>
      <search_type>sthpw/login</search_type>
      <view>table</view>
      <expression>@SOBJECT(sthpw/login)</expression>
    </display>
  </element>
</login>
</config>
```

The widget config views determine how the `TableLayoutWdg` draws itself. There are a few custom attributes that a view can define. The view can define many parts of how the `TableLayoutWdg` is displayed. The following hides the "insert" button and makes each of the cells non-editable. These attributes are useful for reports which are generally not editable.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <simple insert='false' edit='false'>
    <element name="preview"/>
    <element name="code"/>
    <element name="name"/>
    <element name="description"/>
  </simple>
</config>
```