Optimizing Your Workflow with Eclipse and FDT

Created By Xavi Beumala, at 2/27/2006 - 6 comments.

Xavi Beumala specializes in Rich Internet Application development, including CMS, E-learning and real time collaboration applications using Flash, J2EE and FMS.

Click to view this author's website.

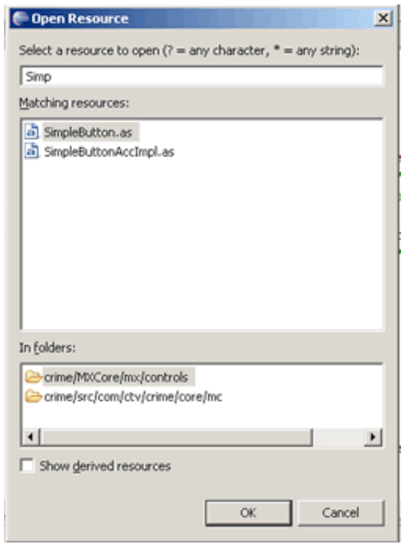
When creating complex Flash applications, we all seem to have developed a number of different workflows, tools and techniques. As an ActionScript developer that uses Eclipse, I've had a lot of experience with ASDT. But I've come to believe that a much better IDE for Flash based enterprise applications is the FDT plug-in for Eclipse. Many people think that using Eclipse implies changing their development workflow by using MTASC and/or swfmill. This is not necessarily the case (although it can greatly reduce your development time and prevent fits of hysteria when compiling).

Eclipse is more than just a tool for highlighting and code completion as many people seem to think. It provides a lot of small and powerful utilities that can help you in your day-to-day work. Some of these tools aren't built directly into Eclipse, but are plug-ins that provides some robust functionality. The FDT plug-in provides many Java-like features for AS based development.

Finding your classes and resources

Flash projects aren't what they once were. Flash based RIA applications are growing in both complexity and size. It's not uncommon for projects to have hundreds of classes and several thousand lines of code. So, we need a way to find classes easily, without having to remember their fully qualified class paths.

When using Eclipse and FDT, you can simply press ***ctrl + shift + R*** to launch the *Find Resource* popup window. This window allows you to write the name of the class (or file) you are looking for and lists the matching files in your project. It's a "find as you type" search, so it's really fast when looking for resources. You can also use wildcards and up- and down- arrows to navigate the results.

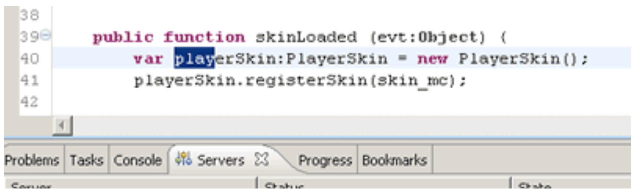


This feature saves you time navigating across your file system, looking for a class or file that may or may not be named what you expected.

FDT also allows you to open .FLA files in Flash from within this window.

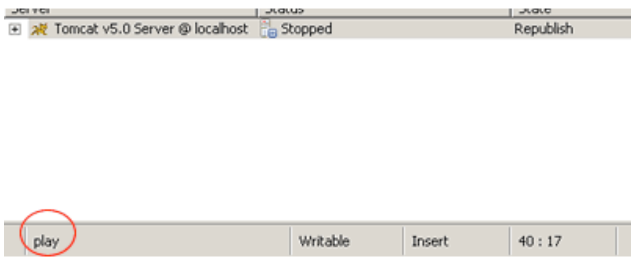
Searching Text

Many times, you'll find that you need to search or search-and-replace actions in your code. You may be looking for a concrete method whilst other times you may just be looking for a string or a regular expression. Sometimes you'll just need to find a single file, but at other times, you may want to search through all of your project source files.

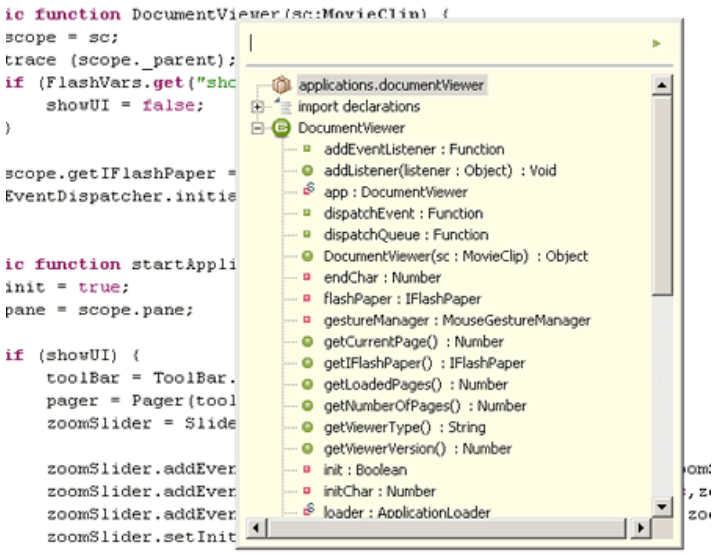


```
public function skinLoaded (evt:Object) {
    var playerSkin:PlayerSkin = new PlayerSkin();
    playerSkin.registerSkin(skin_mc);
```

Depending on your needs, Eclipse provides several ways to accomplish this:

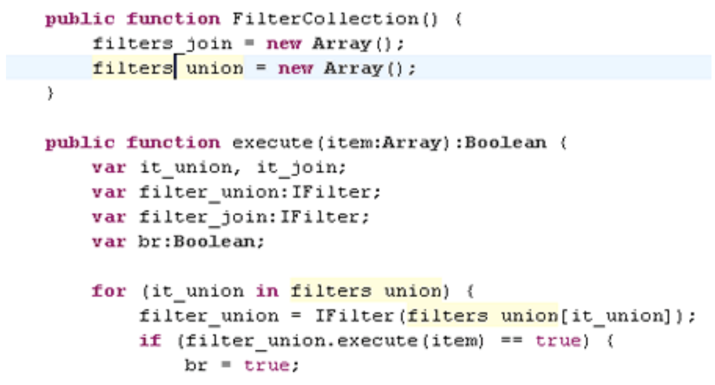


- Searching in different files: **ctrl + H** Type the text or regular expression you're looking for and the pattern of the files you want to look inside. You can choose to search inside a single working set, a single project, selected resources or a full workspace.
- Searching in one file
  - Traditional find/replace **ctrl + F** This window allows you to find/replace text. Once you close this window you can continue searching the same string by pressing **ctrl + K** (next) or **ctrl + shift + K** (previous).
  - Incremental search **ctrl + J** This is very handy and allows you to find text without opening a new window. When you hit the shortcut you can start typing the text you're searching. The first occurrence of whatever you are typing will be highlighted. If you want to repeat the search just hit ctrl + J again (Forward search) or **ctrl + shift + J** (backward search).
  - Many times you'll be searching in a file for a method- or property-declaration. The Outline View gives us a quick insight into a class, but there is a faster way to find it (I can't take my hands off the keyboard). Instead, you can hit **ctrl + O**. Once again, you write the name of the method or property you are looking for and a "find as you type" algorithm will highlight it (just hit enter). This option uses the same configuration as the outline view; so for example, hiding properties in the outline view will also hide them in this popup.
  - The last version of the FDT plug-in includes the capability to *Mark Occurrences*. When placing the cursor caret above any method, property or data type all occurrences of it in the same file will be highlighted.



```
ic function DocumentViewer(sc:MovieClip) {
scope = sc;
trace (scope._parent);
if (FlashVars.get("shc
    showUI = false;
}

scope.getIFlashPaper =
EventDispatcher.initis

ic function startAppli
init = true;
pane = scope.pane;

if (showUI) {
    toolBar = ToolBar.
    pager = Pager(tool
    zoomSlider = Slide

    zoomSlider.addEver
    zoomSlider.addEver
    zoomSlider.addEver
    zoomSlider.setInit
```

```
applications.documentViewer
import declarations
DocumentViewer
    addEventListener : Function
    addListener(listener : Object) : Void
    app : DocumentViewer
    dispatchEvent : Function
    dispatchQueue : Function
    DocumentViewer(sc : MovieClip) : Object
    endChar : Number
    flashPaper : IFlashPaper
    gestureManager : MouseGestureManager
    getCurrentPage() : Number
    getIFlashPaper() : IFlashPaper
    getLoadedPages() : Number
    getNumberOfPages() : Number
    getViewerType() : String
    getViewerVersion() : Number
    init : Boolean
    initChar : Number
    loader : ApplicationLoader
```

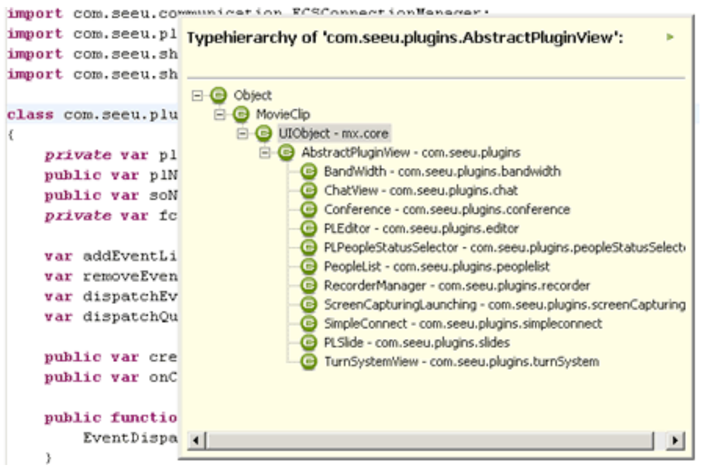


```
public function FilterCollection() {
    filters_join = new Array();
    filters_union = new Array();
}

public function execute(item:Array):Boolean {
    var it_union, it_join;
    var filter_union:IFilter;
    var filter_join:IFilter;
    var br:Boolean;

    for (it_union in filters_union) {
        filter_union = IFilter(filters_union[it_union]);
        if (filter_union.execute(item) == true) {
            br = true;
```

Navigating through your code

Remember these shortcuts ***F3, F4 and F5***. You'll be using them all the time. They are available on the editor view. If you're working on a class, place the cursor caret in a property or method invocation. By hitting ***F3*** the caret will be moved to the line where this property or method is declared. In FDT 1.0.6 you can also hit control key and click on the property or method, it'll have the same effect. Hitting ***F4*** will open the class type of the property. You can also open the class type by placing the caret directly on the class type itself and pressing ***F4***. Pressing ***F5*** will open the super class.

When working with large inheritance chains, it's quite difficult to remember all super classes and implemented Interfaces.



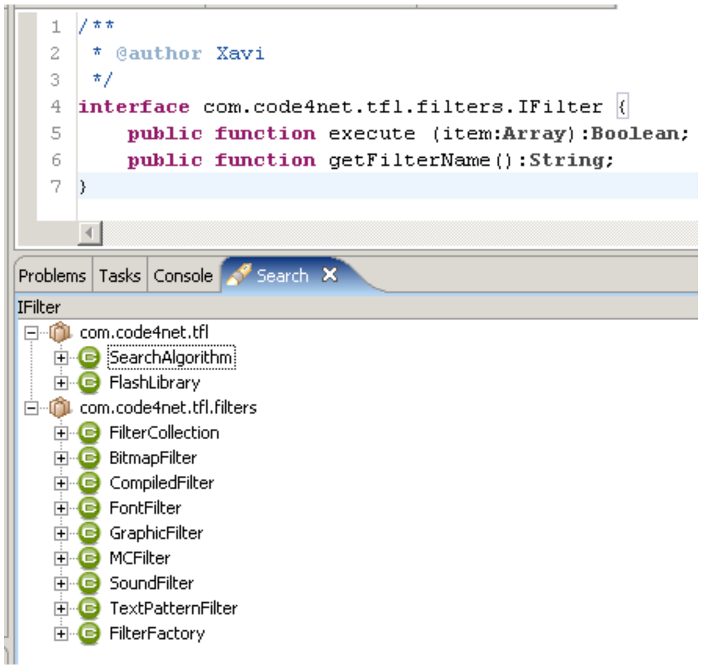
```
import com.seeu.communication.FCSConnectionManager;
import com.seeu.pl
import com.seeu.sh
import com.seeu.sh

class com.seeu.plu
{
    private var pl
    public var plN
    public var soN
    private var fc

    var addEventLi
    var removeEven
    var dispatchEv
    var dispatchQu

    public var cre
    public var onC

    public functio
        EventDispa
    }
```

Typehierarchy of 'com.seeu.plugins.AbstractPluginView':

```
Object
  MovieClip
    UIObject - mx.core
      AbstractPluginView - com.seeu.plugins
        BandWidth - com.seeu.plugins.bandwidth
        ChatView - com.seeu.plugins.chat
        Conference - com.seeu.plugins.conference
        PLEditor - com.seeu.plugins.editor
        PLPeopleStatusSelector - com.seeu.plugins.peopleStatusSelect
        PeopleList - com.seeu.plugins.peoplelist
        RecorderManager - com.seeu.plugins.recorder
        ScreenCapturingLaunching - com.seeu.plugins.screenCapturing
        SimpleConnect - com.seeu.plugins.simpleconnect
        PLSlide - com.seeu.plugins.slides
        TurnSystemView - com.seeu.plugins.turnSystem
```

A way to visually see this chain is with the Type Hierarchy view. You can open this contextual popup by hitting ***ctrl + T***, you'll see the super classes of the currently opened class as well as the sub classes (classes which inherit from the current opened class). This is also quite useful in refactoring processes.

Type Hierarchy offers you a way to navigate your code on a bottom-up way. However if you want to see your top-down class relations or find which other classes reference your current class, you can use the new *Search References* feature by hitting ***ctrl + R***.



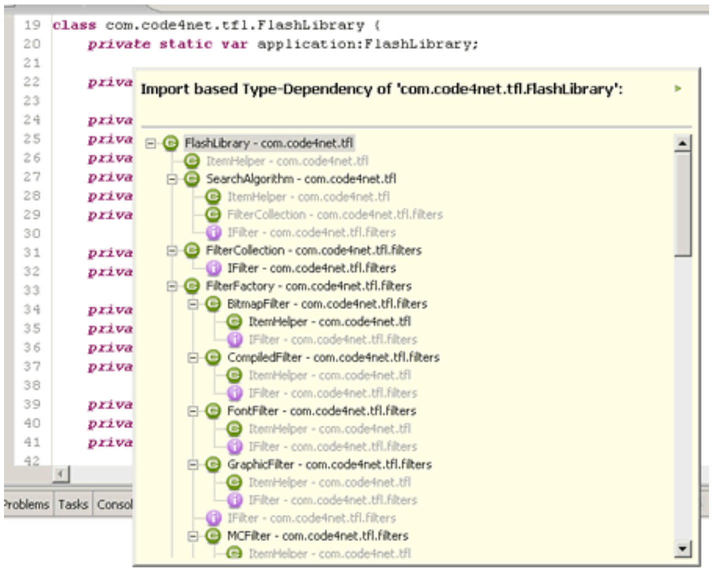
```
/**
 * @author Xavi
 */
interface com.code4net.tfl.filters.IFilter {
    public function execute (item:Array):Boolean;
    public function getFilterName():String;
}
```

Problems | Tasks | Console | Search

```
IFilter
⊟ com.code4net.tfl
    ⊞ SearchAlgorithm
    ⊞ FlashLibrary
⊟ com.code4net.tfl.filters
    ⊞ FilterCollection
    ⊞ BitmapFilter
    ⊞ CompiledFilter
    ⊞ FontFilter
    ⊞ GraphicFilter
    ⊞ MCFilter
    ⊞ SoundFilter
    ⊞ TextPatternFilter
    ⊞ FilterFactory
```

Like many other editors, ***ctrl + L*** will prompt you with a line number to go to. This is useful when debugging.

One really handy thing while developing is the ability to get a quick view of the class being edited. As I've said before, quick outline view is a powerful way to get it, but sometimes it's not enough. If you don't need a schema of the methods and/or properties of your class, but need to see a class's dependency in a hierarchical and graphical way, then *Type Dependency* view is a great choice. You can access it by pressing ***ctrl + U*** while editing a file in the editor.



```
class com.code4net.tfl.FlashLibrary {
    private static var application:FlashLibrary;

    priva

    priva
    priva
    priva
    priva
    priva
    priva

    priva
    priva

    priva
    priva
    priva
    priva

    priva
    priva
    priva
```

Problems | Tasks | Consol

Import based Type-Dependency of 'com.code4net.tfl.FlashLibrary':

```
⊟ FlashLibrary - com.code4net.tfl
    ItemHelper - com.code4net.tfl
  ⊟ SearchAlgorithm - com.code4net.tfl
      ItemHelper - com.code4net.tfl
      FilterCollection - com.code4net.tfl.filters
      IFilter - com.code4net.tfl.filters
  ⊟ FilterCollection - com.code4net.tfl.filters
      IFilter - com.code4net.tfl.filters
  ⊟ FilterFactory - com.code4net.tfl.filters
    ⊟ BitmapFilter - com.code4net.tfl.filters
        ItemHelper - com.code4net.tfl
        IFilter - com.code4net.tfl.filters
    ⊟ CompiledFilter - com.code4net.tfl.filters
        ItemHelper - com.code4net.tfl
        IFilter - com.code4net.tfl.filters
    ⊟ FontFilter - com.code4net.tfl.filters
        ItemHelper - com.code4net.tfl
        IFilter - com.code4net.tfl.filters
    ⊟ GraphicFilter - com.code4net.tfl.filters
        ItemHelper - com.code4net.tfl
        IFilter - com.code4net.tfl.filters
      IFilter - com.code4net.tfl.filters
    ⊟ MCFilter - com.code4net.tfl.filters
        ItemHelper - com.code4net.tfl
```

Editing code related shortcuts

Many times, you'll find yourself cutting and pasting lines of code up and down in classes- placing them in different methods or changing the order of execution between lines. This is really easy to do, but it's faster if you use the move features. Select some lines of code and hit ***alt + upArrow*** or ***alt + downArrow***. The selected code will be swapped with the previous or following lines.

If you want to delete a line, you can do so by hitting ***ctrl + D*** which is much faster than pressing END, ctrl + shift + home and then DEL. You can also duplicate a line by pressing ***ctrl + D***.

shift + ENTER or ***shift + ctrl + ENTER*** allows you to easily add an indented line after or before the line where the cursor is placed.

If you want to comment out some lines of code just select them and hit ***ctrl + 7***

When you're reviewing your code or solving any bug you most probably focus on specific methods. To concentrate more on these methods you can fold the others by pressing ***ctrl + numericPad -*** or ***ctrl + numericPad +*** to unfold it.

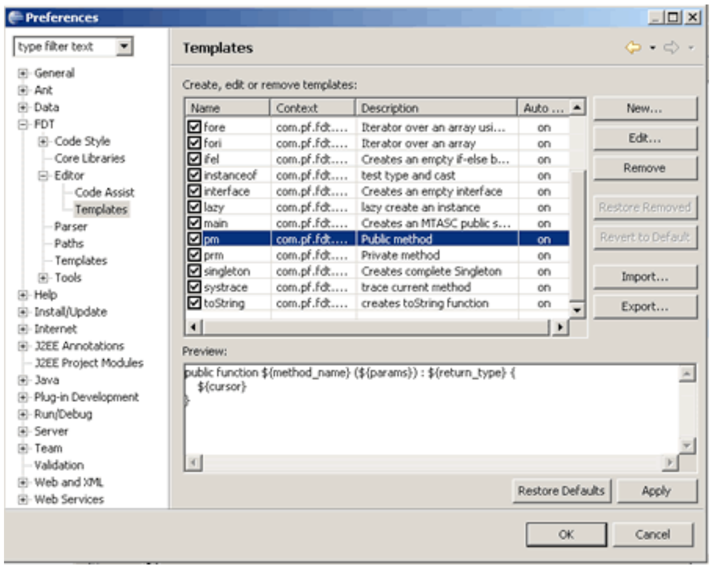
Generating code

Often you'll have to write bits of code over and over again (skeleton). For example whenever you write a couple of new methods, everything in them is the same except for the name of the method, the parameters and the return type (and obviously the stuff inside). To avoid always having to repeat the same thing, you can make use of templates. A template is a code snippet you can insert anywhere by simply typing the name of your snippet and pressing ***ctrl + space***. Eclipse then replaces the name you've typed with the corresponding code (which can be parameterized). To add a new template, go to Window -> Preferences -> FDT -> Editor -> Templates. There you will see all available templates (default templates are really useful!). Let's add a new one to insert a new public method.

Name is the name of the snippet (what you'll type to be replaced with the snippet). My template is named *pm* (PublicMethod). The pattern is the code that will replace the name:

```
public function ${method_name}(${method_params}):${return_type} {${cursor}}
```

Now go to the a file and type *pm* ***ctrl + space*** to insert the template. By hitting the tab key, the cursor will move between the defined insert parameters.



Create, edit or remove templates:

Name	Context	Description	Auto ...
☑ fore	com.pf.fdt....	Iterator over an array usi...	on
☑ fori	com.pf.fdt....	Iterator over an array	on
☑ ifel	com.pf.fdt....	Creates an empty if-else b...	on
☑ instanceof	com.pf.fdt....	test type and cast	on
☑ interface	com.pf.fdt....	Creates an empty interface	on
☑ lazy	com.pf.fdt....	lazy create an instance	on
☑ main	com.pf.fdt....	Creates an MTASC public s...	on
☑ pm	com.pf.fdt....	Public method	on
☑ prm	com.pf.fdt....	Private method	on
☑ singleton	com.pf.fdt....	Creates complete Singleton	on
☑ systrace	com.pf.fdt....	trace current method	on
☑ toString	com.pf.fdt....	creates toString function	on

Preview:

```
public function ${method_name} (${params}) : ${return_type} {
   ${cursor}
}
```

This can seem like a silly thing, but believe me, it really saves a lot of time and provides safety from typing errors.

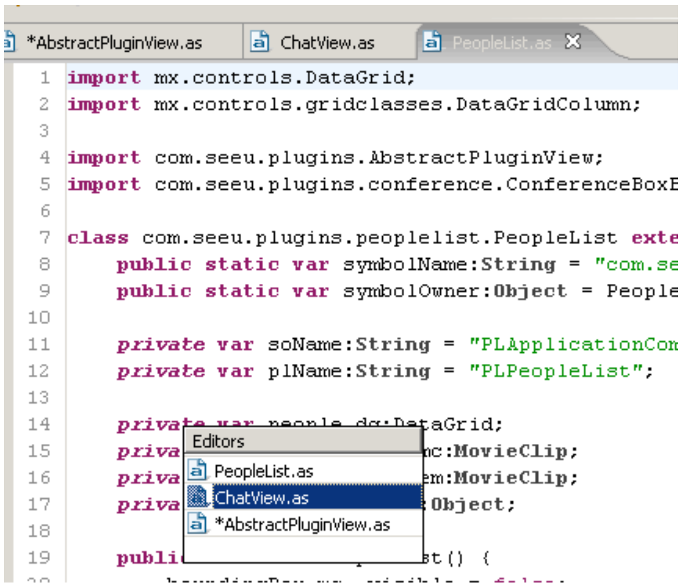
You can also create code another way, but this time from scratch, by using wizards. If you select a package on Flash Explorer view and right click, you can choose new Class, new Interface and so on. There you will be able to select a super class or a super Interface as well as choose class modifiers.

One thing that I can't live without is code auto completion. Just type the first letters of a class, variable or method and hit ***ctrl + space***, and viola! If needed it also will include an import statement.

During the development and refactoring process, many classes are moved, instances are removed, but we always forget about imports. Hitting ***ctrl + shift + O*** will clean unnecessary import statements for us. This works like a charm in an open file, but you can also apply it to a whole project or to a package. Just select the package or folder you want to clean up in the Flash Explorer, right click source -> Organize Imports.

Editing several files

It's not unusual to have several files open at the same time. Moving and finding them in the editor can be extremely annoying. A way to move between open files is by using ***ctrl + Page Down*** and ***ctrl + Page Up***, which will show the previous or following opened file. Another way to do this is with ***ctrl + F6***.



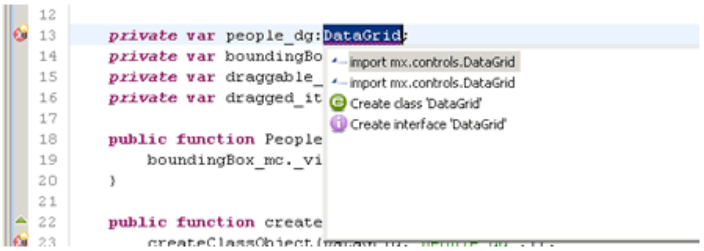
```
import mx.controls.DataGrid;
import mx.controls.gridclasses.DataGridColumn;

import com.seeu.plugins.AbstractPluginView;
import com.seeu.plugins.conference.ConferenceBoxE

class com.seeu.plugins.peoplelist.PeopleList exte
    public static var symbolName:String = "com.se
    public static var symbolOwner:Object = People

    private var soName:String = "PLApplicationCon
    private var plName:String = "PLPeopleList";

    private var people_dg:DataGrid;
    priva                        mc:MovieClip;
    priva                        em:MovieClip;
    priva                        Object;

    publi                        st() {
```

alt + leftArrow and ***alt + rightArrow*** are also incredibly useful. They work like the previous and next buttons in a browser letting you navigate to the previously opened files. This provides you with a history of your steps between files.

Although it's considered best practice to write lines of code that do not exceed 80 characters as maximum, a lot of the time this isn't possible and horizontal scrolls appear. In this case you can maximize the editor by pressing ***ctrl + M*** or ***ctrl + numericPadENTER***.

Fixing problems

There are several common errors that people always encounter whilst coding (forgotten imports, non-existent methods, typing errors, etc.). Sometimes FDT can solve these for you. When a hint is available a bulb will be shown next to the line number. If you hit ***ctrl + 1*** a list of possible fixes will be shown.


import mx.controls.DataGrid
import mx.controls.DataGrid
Create class 'DataGrid'
Create interface 'DataGrid'

```
    private var people_dg:DataGrid;
    private var boundingBo
    private var draggable_
    private var dragged_it

    public function People
        boundingBox_mc._vi
    }

    public function create
```

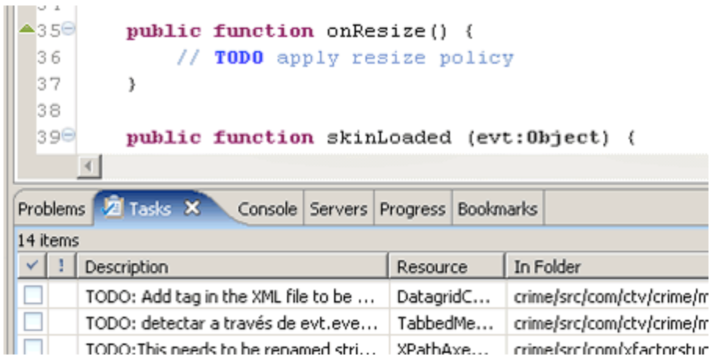
The same way the bulb is shown next to the line numbers, other icons can also be shown as visual aids (formerly annotations). For example: a red cross indicates an error in the code; a green triangle means the method is overriding a super class method; a violet triangle on an interface indicates the method is implementing an interface method; a portfolio-type icon indicates a TODO task; a blue strip is a bookmark; etc…

When editing a file you'll know by annotations if your code has errors; however, with the error view you can see every warning and error in the entire project.

Adding Tasks

When working in a team, annotations in the code are really useful. You can insert comment tasks using TODO, or FIXME. These tasks will be listed on the tasks panel. The following might be a bug in FDT (not sure), but in order to refresh the tasks, you must first clean your project to force reparsing.

When working with a CVS or SVN system, all these notes are shared among users so these notes are placed in the code.

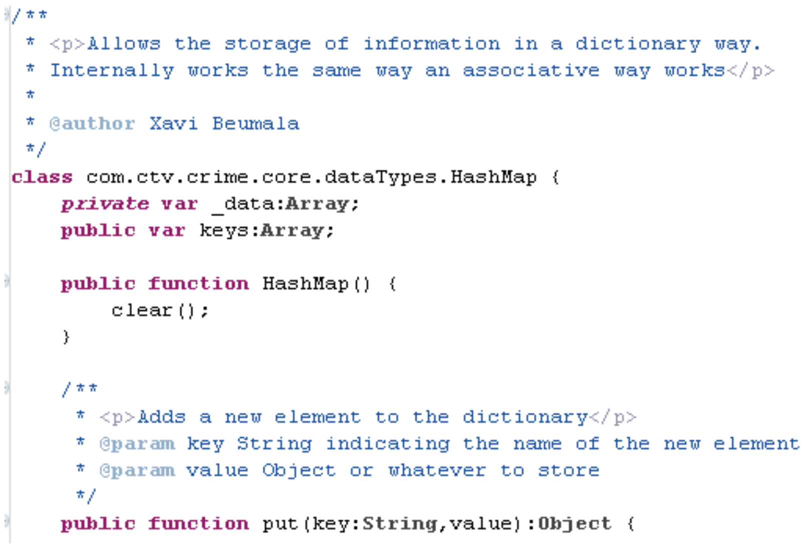


```
    public function onResize() {
        // TODO apply resize policy
    }

    public function skinLoaded (evt:Object) {
```

✓	!	Description	Resource	In Folder
☐		TODO: Add tag in the XML file to be …	DatagridC…	crime/src/com/ctv/crime/m
☐		TODO: detectar a través de evt.eve…	TabbedMe…	crime/src/com/ctv/crime/m
☐		TODO: This needs to be renamed stri…	XPathAxe…	crime/src/com/xfactorstuc

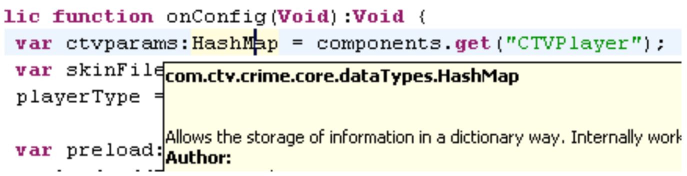
Comments and help

Documenting your code is a best practice, and FDT/Eclipse makes this easier by providing a javaDoc style system. When you write comments between /* and */, you can use auto completion for javaDoc style help items. You can then use any of the available parsers to generate the documentation. Personally, I like **as2api** but there are **others**.



```
/**
 * <p>Allows the storage of information in a dictionary way.
 * Internally works the same way an associative way works</p>
 *
 * @author Xavi Beumala
 */
class com.ctv.crime.core.dataTypes.HashMap {
    private var _data:Array;
    public var keys:Array;

    public function HashMap() {
        clear();
    }

    /**
     * <p>Adds a new element to the dictionary</p>
     * @param key String indicating the name of the new element
     * @param value Object or whatever to store
     */
    public function put(key:String,value):Object {
```

Documentation is an important thing to have in mind. When developing reusable classes, application cores or frameworks you can remember for maybe three months how your class worked: method parameters, public properties, etc. But a lot of time, you'll forget the details, and struggle if you have to use other developer's classes. One solution is to open the class file and read through your code. This can be an "easy" task if it's simple code, but it's not always so simple. Instead, if using Javadoc, you can roll over a method and the javaDoc will pop up.



```
lic function onConfig(Void):Void {
 var ctvparams:HashMap = components.get("CTVPlayer");
 var skinFile
 playerType =

 var preload:
```

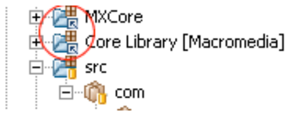
com.ctv.crime.core.dataTypes.HashMap

Allows the storage of information in a dictionary way. Internally work
Author:

This popup also appears if you roll over a code folded comment as well as properties, methods and classes.

Linked Libraries

At work, I prefer to create a new Eclipse project for each new project I start. But I also have a common shared project where I can access my custom core classes. These are reusable classes, helpers, modules and components that I use on a regular basis in all of my projects.



When using these classes from other projects, auto completion doesn't work by itself (this is obvious because it doesn't find the classes I'm referring to). One possible solution is to copy all of these files to each new project. But this would be really untidy and would duplicate a lot of code. Plus it would prevent you from being able to solve bugs in the core framework.

Linked libraries come in handy when you're in this situation. You can reference your classes without embedding them in your project.

To add a linked library, just right-click in your project new -> New Linked Library. Then choose your source folder and you're done!

Building your project

The FDT plug-in allows you to build your project directly with Flash or MTASC. To accomplish this, you have to configure a new builder and set up some parameters. Just right-click on your project and select Run As -> Run… You'll then see an option called "FDT - Flash Support" in the left menu. Double click on it and a new builder will be created. There you'll be able to choose the project that you want to build, as well as the FLA file. Once this is done, you can publish your project by clicking on the run button. Or use ***ctrl + F11***.

On the other hand if you prefer MTASC, you can "FDT - MTASC support" and launch your project with the same shortcut.

Personally, I don't like this way of compiling with MTASC. The configuration isn't saved in a file so you can't share the build process on a CVS system, which complicates the workflow in the team. Instead I use ant based tasks. See **Carlos Rovira's article about using ANT** in ActionScript projects.

Exporting Intrinsic Classes

If you need to deliver a public API of your work so others can use it, it's common to package an SWC file or generate the needed intrinsic classes. FDT can generate your project intrinsic classes for you. Open your project, and right click on it. Choose export and *FileSystem with converter to intrinsic AS2-classes*. The whole structure of your project will be generated but instead of having your working classes, you'll have their corresponding intrinsic ones.

Virtual machine params

Eclipse can be really slow and memory intensive. So if you want to work with agility you must have a lot of available memory. Personally I work with 1.25GB but I've also been running it on a laptop with just 512GB. If you don't have enough memory, you can try to assign some more memory to the JVM. Just add this parameter to your Eclipse call ***eclipse.exe -vmargs -Xmx256M***

Conclusion

Eclipse and FDT bring you a host of tools, utilities and shortcuts to improve and speed up your development process. They also improve the collaboration environment within your development team. Maybe Eclipse is not well suited for small and non-enterprise projects, but believe me when I say that I now develop around 30% faster than before. Although FDT is quite mature, I think it would be awesome if the next release had **basic refactors**. After seeing the last release (1.0.6.1) this is the only thing I can think of that may be missing!!

del.icio.us || Digg It || Technorati

Reader Comments

1. Chris Bizzell Replied:
   (2/28/2006 At 11:43 PM)

   Great article - thanks for putting this together.

2. Carlo Blatz Replied:
   (3/1/2006 At 11:28 AM)

   Woha - thumbs up! Some more Links:

   FDT MainPage:
   http://fdt.powerflasher.com

   FDT Forum:
   http://fdt.powerflasher.com/forum

   More usefull Links:
   http://fdt.powerflasher.com/links

3. Tay Ray Chuan Replied:
   (3/2/2006 At 7:10 AM)

   Hi,
   there is a mistake with your keyboard shortcut for duplicating a line: it should be Ctrl-Shift-D.

   Alternatively, one can use Ctrl-Alt-Down/Up.

   Also, the unit for memory size for your laptop is incorrect: it should be "MB" instead of "GB".

4. eric socolofsky Replied:
   (3/5/2006 At 11:43 PM)

   i looked at FDT's features a while back and was wowed. i've used eclipse for java before and loved it, so i was excited to try FAME. but it's kind of a pain in the ass getting everything set up, and i found no good cross-platform (xp<>osx) solutions. then along came FDT.

   i downloaded the trial and gave it a shot. and very quickly became very disappointed. FDT retails for 200 euro, but despite the not-at-all-negligible price point, it comes with:

   - no structured documentation AT ALL, and
   - no built-in trace/debugging feature.

   how can they sell a program for that much that is not basically plug-and-play? and how can they sell a program for that much that is neither plug and play *nor* documented?

ridiculous, as far as i'm concerned. i strongly recommend against spending money on a slightly-improved version of an existing open-source, FREE software (FAME).

-e

5. **Felix Raab** Replied:
(3/13/2006 At 8:46 AM)

Hi Eric,

I think the price is justifiable if you take into consideration how long it took the author to develop this plugin (as far as I know it was a one-man project). Regarding the docs: There are many videos on the fdt website that show you the most important features (which I personally think is even better than text docs)...and at the latest after reading this article you should know all the useful shortcuts ;). By the way I've provided a sheet containing the shortcuts mentioned in this article **here**.

6. josh Replied:
(3/15/2007 At 4:10 PM)

Just a note: the argument you're passing to the jvm specifies the MAXIMUM amount of ram it'll allocate, which is clearly not what you want here.

You want -Xms256M, not -Xmx256M

Login to post your comments. If you do not have an account with us please **Register**.