

**Consolidation Consultancy Limited**  
**Hyperion Financial Management (HFM)**

**Hints and Tips**

Version: 1.58

Date: 20 October 2009

Author: David Crawford

# Preface

## Note

This document is work in progress and is not finished. Some things need changed and some parts normally indicated by [] have not been finished. Hopefully you will find it useful in its part-finished state.

## About Me

My name is David Crawford I am an accountant with many years' experience of implementing and supporting budgeting and consolidation software. I have developed IFRS, statutory and management systems for many blue chips companies. I now mainly work with Hyperion Financial Management. If you have any projects you need help with please contact me.

## Contacting Me

Landline: +44 (0)20 8393 5565  
Mobile: +44 (0)7761 823795  
Address: Consolidation Consultancy Limited  
72C Reigate Road  
Epsom  
KT17 3DT  
United Kingdom

## Introduction

The idea of this document is to disseminate some of the good ideas I have had and come across while working with Hyperion Financial Management (HFM). Some of the tips may seem very obvious but they are all based on real issues I have encountered while consulting.

# Principals

Principals to apply when developing Hyperion Financial Management (HFM) applications are:

## Defaults

### Principal

Accept the default settings.

### Justification

- Simplifies build
- Simplifies maintenance
- Saves money

### Examples

- Software installation folders. It might seem a good idea to install software in C:\Program Files\Hyperion instead of C:\Hyperion but it is not. The default location is always better.
- Form fonts, colors and so on.
- Report fonts, margins and so on

## DRY (Don't Repeat Yourself)

### Principal

Repeat as little data, metadata and rules as is practical and consistent with KISS.

### Justification

- Principal applies to all computer systems.
- Not repeating metadata and rules makes systems more reliable and easier to maintain.
- No repeating data means no reconciliation.

### Examples

General examples of this principal are:

- Normalisation of relational databases; and
- Using subroutines, functions and objects to hold common code.

These examples have close parallels in Hyperion Financial Management (HFM) which are explained in the following sections of this document:

- Analysis Accounts
- Split Subroutines
- Statutory Accounts Enrichment

## KISS (Keep It Simple)

### Principal

Make applications as simple as possible.

### Justification

- Improves usability
- Simplifies building and maintenance
- Reduces training

- Saves money

### Examples

- Simple formatting of reports

The following are normally too complex for most Hyperion Financial Management (HFM) applications.

- Automating associate and minority interest accounting.
- Automating consolidation eliminations, investment in subsidiaries and so forth. It is just easier for accountants to write journals

## Reporting

### Principal

All core reports should be in Hyperion Financial Reports. Smart View should only be used for user generated reports.

### Justification

- Packs can be produced at the click of a button.
- Maintenance is reduced as new members are automatically added to Hyperion Financial Reports
- Support is reduced as all users run the same report.

## Responsibilities

### Principal

The responsibilities of finance and system teams should be clearly defined. The finance team should be responsible for data and the system team should be responsible for the system.

### Justification

Clearly defined responsibilities let everyone focus on what they are good at.

### Examples

<b>Team</b>	<b>Responsibility</b>
Finance Team	Data Journals Mappings Process Control Smart View
Systems Team	Metadata Rules Documents (except user grids) Hyperion Reports Security

# Accounts

## Analysis Accounts

### Tip

Analysis accounts containing financial data that is part of the trial balance should be included in the main balancing hierarchies, normally profit and loss and balance sheet.

### Justification

Some times analysis accounts are configured as follows:

```
Profit
  |--OperatingCost
    |--GrossProfit
      |--Turnover
        |--COS
TurnoverAnalysis
  |--Product A
  |--Product B
  |--Product C
```

The total from `TurnoverAnalysis` is transferred to `Turnover` by a rule. This breaks the DRY principal as the value for `Turnover` exists twice in the database. The same functionality can be more simply implemented as follows:

```
Profit
  |--OperatingCost
    |--GrossProfit
      |--Turnover
        |--Product A
        |--Product B
        |--Product C
        |--COS
```

In older versions of Hyperion Financial Management (HFM) this may not be possible if analysis accounts are collected at a different time in a different Scenario. While the example about is for the accounts hierarchy to principal applies to all hierarchies

## Balance Sheet Accounts

### Tip

In statutory IFRS style applications all balance sheet accounts should have a movement table in a custom dimension. For accounts for which movements do not need to be disclosed the movements should be:

Acquisition of subsidiaries  
Movements  
Disposal of subsidiaries  
Foreign exchange

### Justification

I have singled out IFRS because IFRS and similar standards (UK GAAP) require the disclosure of movement information that is not required by US GAAP. I think US GAAP applications would also benefit from all accounts having movement tables but as I have not built one I would not like to suggest it as a standard. Does anyone have an opinion?

Giving all balance sheet accounts movement tables has the following advantages:

- If all balance sheet accounts do not roll forward journals posted to accounts that do not roll forward will not balance in future years.
- Makes cashflows easier to calculate as acquisition and disposal balance sheets are easily identified.

## Descriptions

### Tip

Descriptions should be sentence case that is first letter capitalised.

### Justification

I would tend to use title case (first letters capitalised) but if you look at published accounts they mostly use sentence case for account descriptions. I think once you get used to it, it looks better and is easier to read.

### Examples

#### Good

Fixed assets

Creditors less than one year

#### Bad

Fixed Assets

Creditors Less Than One Year

## IsICP

### Tip

IsICP should be set to R for intercompany accounts.

### Justification

Prevents entities selecting themselves as partners for intercompany input.

## Labels

See Metadata section.

## Promoting Accounts

### Tip

When promoting an account from base to parent a new parent account should be created. The existing account should be made no input.

### Justification

Base accounts cannot be changed to parent accounts because their new children would overwrite their data.

### Example

#### Before

```
1200 - Consultants cost
```

#### After

```
1201 - Consultants cost
|--1210 - Consultants PWC cost
|--1220 - Consultants Accenture cost
|--1220 - Consultants other cost
`--1200 - Consultants cost (Closed)
```

# Calendars

## Period Names

### Tip

When creating \*.per files month labels should be three letters. The month description should be the full month name.

### Justification

Makes short and long descriptions for months available in Hyperion Reports.

### Example

<u>Label</u>	<u>Description</u>
Jan	January
Feb	February

# General

## Data Migration

### Tip

Do not use Hyperion tools (HAL or FDM/ FDQM) to migrate data from an old system to Hyperion Financial Management (HFM). Try using desktop tools like MS Access they are usually quicker and easier.

### Justification

I think the advantages of desktop tools are:

- Shorter lead-time for installation. MS Access is often already installed on workstations.
- Skills are readily available.

I think the disadvantages of Hyperion tools are:

- To complex and slow for data migration.
- FDM is optimised for loading single periods and has limited functionality for loading multiple periods.
- Skills are less readily available.

# Hyperion Reports

## Advanced Member Selection

### Tip

When creating a report of the following hierarchy:

```
IE1448 Net surplus before Exceptional Items
|--IE1010 EBITDA
|--IE1450 Depreciation
|--IE1455 Net interest
`--IE1475 Other
```

The following rows can be specified to obtain the desired formatting.

		A
		Actual
1	<b>IE1010</b>	#
2	IE1450	#
3	IE1455	#
4	IE1475	#
5	<b>IE1448</b>	#

This is not dynamic and if an extra child is added to IE1448 Net surplus before exceptional items it needs to be manually added to the grid. However, the following grid that uses advanced member selection is completely dynamic.

		A
		Actual
1	<b>IE1010</b>	#
2	Children of IE1448 AND NOT IE1010	#
3	<b>IE1448</b>	#

Advanced member selection is activated using the View button on the Members selection dialog box. See *Selecting Multiple Members Based on Criteria* in the *Hyperion Reports* manual for more information.

## Borders and Formatting

### Tip

Simple borders and formatting should be used on reports.

### Justification

- Complex formatting is difficult and time consuming to create and maintain.
- Complex formatting can confuse reports.
- Complex formatting often looks messy as most report writers do not have graphics training.

## Change Control

### Tip

- New or revised reports should be created in a sub folder named `_New` within the folder containing the existing report. Only the report writer and the users responsible for reviewing the report should have access to this folder.
- When a report is approved the old report should be copied to a sub folder named `_Old` within the folder containing the old report. The new report should then be copied to the main folder.

### Justification

Using a development report server is often unnecessary and impractical as development systems often do not have full sets of data. This procedure provides sufficient safeguard to avoid accidental changes and allows rollback.

## Column Widths

### Tip

Standard column widths should be defined for reports. Suggested column widths are:

Data	70	This is the default column width and seems to work well.
Formula	70	Ditto
Separators	15	Narrower columns are difficult to select.

### Justification

Helps reports have a consistent appearance.

## Designer

### Tip

- For single grid reports the entire grid should be visible in the report designer screen.
- For single grid reports the grid should be made as large as possible.

### Justification

- Making the grid fit on the screen means you have to use one set of scroll bars to edit the grid. Grids will autosize to show all the members selected.
- Making grids as large as possible makes the most information possible visible to the report writer.

## Dimensions

### Tip

Set dimensions that do not change on a grid with the grid POV.

### Justification

Putting dimensions which do not change on a grid m

## Folder Hierarchies

### Tip

Folder hierarchies should have the fewest possible levels and be limited to a maximum of three levels.

### Justification

[]

## Folder Names

### Tip

Folder names should be three characters. Normally the characters should be letters though numbers could be used if it is logical and improves usability.

### Justification

Three character names provide a short reference for folders. The folder description can be used to provide more information. Folder names are used in the report names standard and using three character codes keeps report names to manageable lengths.

### Examples

ACT	Actuals
FOR	Forecast
PLN	Plan

### See Also

Report Names

## Font Size

### Tip

Fonts smaller than 8pt should not be used.

### Justification

- 8pt is the default font size for Grids, presumably for a reason.
- Smaller fonts are difficult to read.
- If a grid does not fit on a page it can be made smaller using the fit to page options **File > Page Setup**.

## General

### Tip

Reports should be as dynamic as is practical

### Justification

Dynamic reports will automatically reflect changes to hierarchies and will not need to be manually updated.

## Grid Dimensions

### Tip

Put dimensions on the grid in the standard dimension order:

Scenario, Year, Period, View, etc

### Justification

□

## Linked Objects

### Tip

Put all linked objects in one folder. Do not group linked objects in subfolders.

### Justification

□

## Margins

### Tip

Standard margins should be defined for reports.

### Justification

Helps reports have a consistent appearance.

## Network Backup

### Tip

All reports from the Reports Repository should be backed up as \*.des files on a network drive.

### Justification

Provides a backup of the live reports in case you accidentally break them.

## Number Formats

### Tip

Use the following prefixes and suffixes:

	<u>Prefix</u>	<u>Suffix</u>
Positive numbers	Space	Space
Negative numbers	(	)

### **Justification**

Lines up numbers like this:

```
(100)
 100
```

Instead of like this:

```
(100)
 100
```

## **Replace With**

### **Tip**

Replace with (Format > Cells > Replace tab) should not be used when text can be displayed using alternatives such as Custom Headings.

### **Justification**

The alternatives to Replace With are easier to use and maintain.

## **Report Names**

### **Tip**

Report names should be the folders that contain the reports joined by dashes followed by a three-digit number. The numbers sequence should leave gaps so new reports can be added.

### **Justification**

- Provides short names for reports.
- The report description can be used to provide a full description for users.
- Ensures every report has a unique name and shows the folders reports are in. This is useful as there is no search facility in Hyperion Reports 7.2.
- The three digit numbers can be used to sort reports in the order they appear in books.

### **Examples**

```
ACT-BOA-110 Actual Board Profit and Loss
ACT-BOA-120 Actual Board Balance Sheet
```

## **Row Heights**

### **Tip**

Standard row heights should be defined for reports. I suggest multiplying the font size by two.

### **Justification**

Standard heights help give reports a consistent appearance. The standard font is 8 and the standard row height is 18. 16 makes the maths easier and saves a little paper.

## **Suppressing Formula Columns**

### **Tip**

Suppress formula columns using the equals suppressed condition in Advanced options:

1. Select the formula column you want to suppress.
2. In Suppression check **Advanced Options**.

3. In Suppression click **Setup**.
4. In the first condition column select **Data Values in Column**.
5. In the second condition column select a data column with the suppression you need. If you do not have one create a extra column give it the suppression you want ands hid it.
6. In the third condition column select =.
7. In the fourth condition column select **Suppressed**.
8. Click **OK**.

### **Justification**

You cannot suppress formula columns dynamically depending on dimensions of the point of view like month. Well you can but sometimes it does not work.

# Infrastructure

## Operating Systems

### Tip

Install Hyperion System 9 exclusively on Windows servers.

### Justification

While some elements of Hyperion System 9 (for example Shared Services) can be installed on Unix operating systems some ( for example Financial Management and FDM) require Windows. Its makes life easier if everything is installed on Windows servers. It means one server administrator can deal with all Hyperion System 9 issues. If one part is installed on Unix servers and others parts on Windows servers you will probably need to involved two administrators to fix problems.

# Metadata

## Labels

### Tip

If you are not using numeric codes most of the standards for Rules variables apply to naming Metadata labels

### See Also

Rules Variable Names

# Rules General

## Commenting

### Tip

Comments should be formatted as follows:

```
'***  
' Calculates cashflows.  
,  
' The first line should be a short summary of what the subroutine does  
' followed by a blank line. These lines should be a more detailed description  
' of the subroutine. Complex descriptions for how a subroutine works should be  
' included here rather than in the body of the subroutine.  
,  
Sub CalculatesCashflow  
  
    '***  
    ' Large sections of code should be commented like this.  
    ,  
  
    ' Small blocks of code should be commented like this.  
  
End Sub
```

See <http://java.sun.com/j2se/javadoc/writingdoccomments/> for authoritative code documentation standards.

### Justification

- Adapts standard commenting style to VBScript.
- '\*\*\* is quicker to create and maintain than the normal VBScript commenting styles.

## Const Statement

### Tip

Use the Const statement to declare constants.

### Justification

Rules files sometimes declare constants as global variables as follows:

```
Dim AllTops  
Dim AllTopsXC1  
  
Sub Calculate()  
  
    AllTops = "C1#AllC1.C2#AllC1.C3#AllC3.C4#AllC4. I#[ICP Top]"  
    AllTopsXC1 = "C2#AllC2.C3#AllC3.C4#AllC4.I#[ICP Top]"
```

The correct way to declare them as constants is:

```
Public Const ALL_TOPS = "C1#AllC1.C2#AllC2.C3#AllC3.C4#AllC4. I#[ICP Top]"  
Public Const ALL_TOPS_XC1 = "C2#AllC2.C3#AllC3.C4#AllC4.I#[ICP Top]"  
  
Sub Calculate()
```

The values of constants created using this syntax cannot be altered.

## Constant Names

### Tip

Constant names should be upper case with words separated by underscores.

### Justification

- Capitalisation is standard practice.
- Underscores make names more readable.

## Examples

```
Public Const ALL_TOPS = _
    ".I#[ICP Top].C1#A11C1.C2#A11C2.C3#A11C3.C4#A11C4"
Public Const ALL_NONES = _
    ".I#[ICP None].C1#[None].C2#[None].C3#[None].C4#[None]"
```

## For Each

### Tip

Where possible use `For Each Member In List` instead of `For i = Lbound(List) To Ubound(List)`.

### Justification

Hyperion Financial Management (HFM) rules sometimes have `For` loops as follows:

```
AccountList = HS.Account.List("A#Register", "[Base]")
For i = Lbound(AccountList) To Ubound(AccountList)
```

```
    'Code in loop
```

```
Next
```

These can be changed to `For Each` loop as follows:

```
AccountList = HS.Account.List("A#Register", "[Base]")
For Each AccountMember In AccountList
```

```
    'Code in loop
```

```
Next
```

`For Each` loops are:

- Simpler
- Easier to read and understand.
- Probably quicker as array length does not have to be calculated for every loop.

## If Then Versus Select Case

### Tip

Generally `If ... End If` should be used. `Select Case... End Select` should only be used where it significantly more concise and improves readability.

### Justification

`If...End If` is more like natural language and normally easier to use than `Select Case...End Select`. However, sometime `Select Case...End Select` produces more readable code and is preferred.

### Examples

```
If CurrentMonth = "Jan" Then
    'Code
ElseIf CurrentMonth = "Feb" Then
    'Code
ElseIf CurrentMonth = "Mar" Then
    'Code
End If
```

Is preferable to:

```
Select CurrentMonth
Case "Jan"
    'Code
Case "Feb"
    'Code
Case "Mar"
    'Code
End Select
```

But

```
Select CurrentMonth
Case "Jan", "Feb", "Mar"
    'Code
End Select
```

Is preferable to

```
If CurrentMonth = "Jan" _
Or CurrentMonth = "Feb" _
Or CurrentMonth = "Mar" Then
    'Code
End If
```

## Indenting

### Tip

Indent code four spaces to show code structure.

### Justification

Visual Basic standard.

## NoInput Subroutine

### Tip

Put NoInput rules in a separate subroutine called MakeNoInput. Suggestions for a better name gratefully accepted.

### Justification

Make it easy to disable NoInput rules if required, for example to load historic data.

## Option Explicit

### Tip

Option Explicit should be used for complex rules files.

### Justification

The Option Explicit statement, requires all variables to explicitly declare using the Dim, Private, Public, or ReDim statements. If you attempt to use an undeclared variable name, an error occurs. Option Explicit prevents incorrectly typing the name of an existing variable and avoids confusion in code where the scope of the variable is not clear.<sup>1</sup>

The following example illustrates affect of the Option Explicit statement.

```
Option Explicit ' Force explicit variable declaration.
Dim MyVar ' Declare variable.
MyInt = 10 ' Undeclared variable generates error.
MyVar = 10 ' Declared variable does not generate error.
```

## Profiling

### Tip

You can check how long sections of code take to run as follows:

```
Dim StartTime, EndTime
StartTime = Timer

'Code you want to test

EndTime = Timer
WriteToFile("Took = " & (EndTime - StartTime))
```

---

<sup>1</sup> Microsoft Windows Script Technologies documentation.

## Split Subroutines

### Tip

Long predefined routines should be split into sections using subroutines.

### Justification

Many applications have Calculate subroutines that are thousands of lines long. This conflicts with programming best practice<sup>2</sup> that requires subroutines to be short and to the point. As the code in Hyperion Financial Management (HFM) rules is often simple lists of HS.Exp() statements I think Hyperion Financial Management (HFM) subroutines can be longer than normal but that applications with subroutines of more than a few hundred lines of code will benefit from being split up. Possible subroutines are:

- CalculateCashflow
- CalculateKPIs

A few of the benefits of splitting long subroutines are:

- Reduces the scope of variables making rules simpler to write and easier to maintain.
- Rules development can be split by subroutine allowing more than one developer to write rules and help avoid the Rules hat problem.
- Allows code to be more easily performance profiled.
- Allows code to be more easily optimised as new code can easily be called in place of the old code to see if performance improves.

## Subroutine Names

### Tip

Subroutines should be named VerbObject, for example CalculateCashflow and GetPriorScenario.

### Justification

Makes code easier to understand.

## VBScript Documentation

### Tip

Download Microsoft Visual Basic Scripting Edition documentation from <http://www.microsoft.com/downloads/>. If you cannot find it there search for *vbscript chm site:microsoft.com* in Google.

### Justification

Contains full documentation of the VBScript language used by Hyperion Financial Management (HFM).

---

<sup>2</sup> Code Complete.

## Rules Variable Names

Please note some of these standards are very much my personal preference. Please do not flame me if you disagree. See Code Complete for extensive discussion of variable naming.

### Avoid Abbreviations

#### Tip

Avoid abbreviating variable names.

#### Justification

- Variable names up to 20 characters do not impair the readability of code<sup>3</sup>.
- Abbreviations make code less like natural English and impair understanding.
- Means not having to ensure abbreviations are consistent.
- Means not having to guess which abbreviation to use, for example Man, Mgt or Mngmt.

#### Examples

<u>Good</u>	<u>Bad</u>
AccountIndex	i
PriorPeriod	PrPrd

### Prefixes

#### Tip

Variable names should not have prefixes.

#### Justification

I think preceding variable names with prefixes makes them and the code they are in less like natural English and therefore more difficult to understand. Generally the variable type should be obvious from the variable name. Also VBScript is not a typed language so type is less important than in other languages

#### Examples

<u>Good</u>	<u>Bad</u>
AccountIndex	intAccount
AccountsList	astrAccount
AccountMember	strAccount

### Standard Suffixes

#### Tip

Use the following standard suffixes for variable names.

<u>Suffix</u>	<u>Use</u>	<u>Example</u>
Count	Holds the number of members in a list or elements in and array.	AccountsCount
Index	Hold the currently member or element of any array in a loop.	AccountIndex
List	Holds a list of members. <sup>4</sup>	AccountsList
Member	Holds a member of a list.	AccountMember

---

<sup>3</sup> Code Complete.

<sup>4</sup> This is a bit verbose I would normally use Account and Accounts but these names conflict with names used by the HS object so I use AccountMember and AccountsList in HFM.

**Justification**

Makes code consistent and therefore easier to read and maintain.

# Scenarios

## Default Frequency

### Tip

All scenarios in an application should have the same default frequency.

### Justification

- Having different frequencies can confuse users.
- Where data is not collected in all periods the unused periods can be blocked with no input rules.

## Forecasts

### Tip

If forecasts are collected every month the current forecasts should be stored in the same scenario every month. If prior forecasts need to be retained the current forecast should be copied to storage scenarios.

### Example

```
ForecastCurrent  
ForecastFeb  
ForecastMar  
:  
ForecastOct  
ForecastNov
```

### Justification

Having the current forecast in the same scenario every month makes reports much easier. The disadvantage of this approach is that reports for previous months cannot be rerun.

# Security

## Custom Dimensions

### Tip

If at all possible security should not be enabled for custom dimensions.

### Justification

- Enabling security for custom dimensions greatly increases the complexity of the security model and is very difficult to maintain. Keeping entity security up to date is normally a challenge.
- In statutory applications access to balance sheet movements can generally be suitably controlled by NoInput rules and by omitting movements from input users' data entry forms.

## Class Names

### Tip

Security class names should be prefixed with a letter indicating what the class is securing followed by an underscore and the label of the member being secured. For example E\_1000 would be the security class for entity 1000.

### Justification

- Using prefixes helps distinguish what the class is securing. Even if you are only securing entities using prefixes allows you to easily add additional security in the future.
- This is a common convention.

## Default Role

### Tip

When using groups to provision users with roles users must be manually provisioned with the Default roles.

### Justification

If users are not provisioned with the Default roles they cannot be provisioned with classes.

## Native Groups

### Tip

Use native groups in Shared Services to provision users with roles and classes. Almost all applications will benefit from using groups for roles and most will benefit from using groups for classes.

### Justification

- Native groups can be maintained by the Shared Services administrator.
- Provisioning a few groups with tasks and assigning users to groups is much easier than provisioning hundreds of users with roles.

## Native Groups Classes

### Tip

Use class groups to provision users with access to divisions where the children have different security classes.

### Justification

- Restricting access to divisions by the individual entities in divisions can be very difficult to configure and maintain.

## Example

ABC Group has the following entities:

```
3000 ABC Group
  |-3010 ABC UK Group
    |   |-3020 UK IT
    |   |-3030 UK Consulting
    |   |-3040 UK Technology
    |   `--3050 UK Overhead
  |-1002 ABC Germany GmbH
  |-1003 ABC USA Ltd
  |-1004 ABC France SA
  |-1005 ABC Italy SA
  `--1006 Consol adjustments
```

To give UK divisional management access to entity 3010 and its children.

1. Create a group HFM\_ManClassesE\_3010.
2. Add the following classes to the group:  
E\_3010  
E\_3020  
E\_3030  
E\_3040  
E\_3050
3. Assign UK divisional users to this group.

## Native Groups Names

### Tip

Name native groups as follows:

<u>Template</u>	<u>For</u>
HFM_<Application>Roles<Name>	Roles
HFM_<Application>Classes<Parent>	Classes

### Justification

- Allows groups for HFM and each HFM application to be easily identified in Shared Services by searching for HFM\* and HFM\_<Application>\*.
- The underscore after HFM is the make the names easier to read.

### Examples

<u>Group</u>	<u>Description</u>
HFM_ManRolesInput	Roles for management application
HFM_ManRolesDivision	
HFM_ManClasses1000	Classes for entities in US
HFM_ManClasses2000	Classes for entities in Europe
HFM_StatRolesInput	Roles for statutory application
HFM_StatRolesDivision	

# Smart View

## General

### Tip

Use functions and not ad-hoc grids for creating reports and saving data to Financial Management.

### Justification

- Functions are more robust than Excel grids
- Data retrieved from Financial Management can be formatted.

## Parameters

### Tip

Link the connection and every dimension of HS functions to a cell.

### Justification

This makes you reports more flexible and easier to maintain.

### Example

[]

## Printing

### Tip

- Use the Excel print area function (**File > Print Area > Set Print Area**) to hide areas you do not want to print.
- Use page break preview (**View > Page Break Preview**) to check you have set the print area correctly.

### Justification

Hides the workings for Smart View formulas and is easier the hiding and unhiding rows and columns.

## Sheets

### Tip

- Put all function parameters (connections and dimensions) on all pages of multi-page workbooks.
- Link parameters that do not change to the first page of the workbook.

### Justification

Parameters which do not change could be linked to the first page of the workbook but the resulting formula syntax `Sheet1!A1` make the HS formulas much longer and more difficult to debug.

## Tools

The following are freeware programs I have found useful when working with Hyperion Financial Management (HFM). They would also be useful for people working with Hyperion Enterprise. I spent hours trying out different programs before selecting these. Hopefully this list will save you going through the same exercise. For a great list of freeware see <http://www.techsupportalert.com>.

### Agent Ransack

They say, Agent Ransack is a free tool for finding files and information on your hard drive fast and efficiently. When searching the contents of files Agent Ransack displays the text found so you can quickly browse the results without having to separately open each file.

I have used Agent Ransack to search through hundreds of Web Data Entry Forms files (\*.wdf) and Reports files (\*.des) to find which use particular accounts.

### IZArc

They say, With a modern easy-to-use interface, IZArc provides support for most compressed and encoded files, as well as access to many powerful features and tools. It allows you to drag and drop files from and to Windows Explorer, create and extract archives directly in Windows Explorer, create multiple archives spanning disks, creating self-extracting archives, repair damaged zip archives, converting from one archive type to another, view and write comments and many more. IZArc has also build-in multilanguage support.

I have used IZArc at organisations that where to mean to put WinZip on every PC.

### PSPad

They say, The universal freeware editor, useful for people who: work with plain text - the editor has a wealth of formatting functions, including a spell checker; create web pages - as a web authoring editor, PSPad contains many unique tools that save your time; want to use a good IDE for their compiler - PSPad catches and parses compiler output, integrates external help files, compares versions and much more.

I have used PSPad to edit \*.rle files. The folding feature lists and functions and subroutines making it easier to navigate rules files with many subroutines.

### Replacem

They say, At its core, ReplaceEm is essentially a text search-and-replace program. However, unlike the search-replace functionality of a standard text editor, ReplaceEm is designed to operate on multiple files at once. And you need not only perform one search-replace operation per file -- you can setup a list of operations to perform. If different groups of files need to have different operations performed on them, this is no problem either. You can also specify a backup file for each file processed just in case the replace operation didn't do exactly what you wanted. See <http://www.fitsoftware.com/replace> for more information.

I have used Replacem to perform bulk changes to \*.des file generated by Hyperion Reports.

### WinMerge

They say, WinMerge is an Open Source visual text file differencing and merging tool for Win32 platforms. It is highly useful for finding what has changed between project versions, and then merging changes between versions. See <http://winmerge.sourceforge.net> for more information.

I have used it to compare Rules files it could also be used to compare Members Lists and to a limited extent HFM \*.app files. See Consolidation Consultancy Hyperion Financial Management (HFM) Utilities documentation for a fuller discussion of comparing HFM \*.app files.