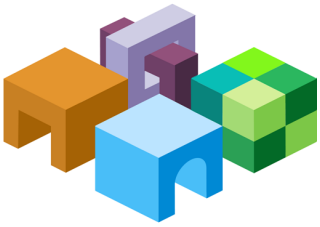


ORACLE® ESSBASE

RELEASE 11.1.1.3

ADDENDUM

ORACLE
ENTERPRISE PERFORMANCE MANAGEMENT SYSTEM

This addendum to the Oracle Essbase documentation set describes, in depth, the new features in Essbase Release 11.1.1.3.

Contents In Brief

Drill-through from Essbase to Oracle Applications 2
FDM Enhancements 51
Smart View Enhancements 52
Calculation Manager Enhancements 52

Drill-through from Essbase to Oracle Applications

Essbase provides URL-based drill-through access from Essbase client reporting interfaces (such as Oracle Hyperion Smart View for Office, Fusion Edition or Oracle Hyperion Financial Reporting, Fusion Edition) to information located on Oracle Enterprise Resource Planning (ERP) applications and Enterprise Performance Management (EPM) applications.

See these topics:

- "Overview of Drill-through to Oracle Applications" on page 2
- "Drill-through URLs" on page 3
- "Creating and Managing Drill-through URLs" on page 4
- "MaxL Statements" on page 5
- "MaxL Definitions" on page 8
- "C Main API Structure" on page 10
- "C Main API Functions" on page 10
- "C Grid API Structure" on page 19
- "C Grid API Function" on page 20
- "Visual Basic API Structure" on page 21
- "Visual Basic API Functions" on page 22
- "Drill-through Visual Basic API Example" on page 28
- "Administration Services" on page 50

Overview of Drill-through to Oracle Applications

This documentation discusses the ability to drill through to information hosted on Oracle ERP and EPM applications, which differs from the concept of drill-through as described in Oracle Essbase Integration Services and Oracle Essbase Studio documentation. In this documentation, drill-through refers to the ability of an Essbase database cell to link to information contained in another Oracle application. In Integration Services and Essbase Studio documentation, drill-through refers to linking a multidimensional database cell to further data; for example, to transaction-level data that is stored in a relational source.

When you deploy an Essbase database using Oracle General Ledger or Oracle Hyperion EPM Architect, Fusion Edition, you use the Essbase API to populate the Essbase database with information about which Essbase database cells are valid regions enabled for drill-through access to the enterprise-reporting applications.

For each Essbase drillable region of an Essbase database, you enable drill-through access by means of a URL. Use the Essbase API to populate the Essbase database with the URL information, as well as the drillable-region information. The URL displays to users of the Essbase client interfaces as a link associated to a cell which provides access to related information hosted by Oracle ERP and EPM applications.

For example, in Smart View, a cell represents actual sales data for Cola in the New York market in January. Color-coding on the cell indicates that there are reports associated with the cell. This particular cell could have multiple links associated through which the user can scroll. Each link is enabled by a URL. When the user clicks on a link, the URL is validated, and a launch page, hosted by the ERP or EPM application, displays in a Web browser.

Drill-through URLs

ERP and EPM applications create a drill-through URL using Essbase. The drill-through URL is stored in the Essbase database file as metadata.

Note: It the responsibility of the Administrators of the ERP and EPM applications to define drill-through definitions and host the Web pages that they wish to use as targets of drill-through URLs.

Drill-through URLs consist of the following components:

- "Drill-through URL Name" on page 3
- "Drill-through URL XML" on page 3
- "List of Drillable Regions " on page 3
- "Level-0 Boolean Flag" on page 4

Drill-through URL Name

The drill-through URL name is an identifier to manage the defined drill-through URL This name can be different from the URL display name visible to the end user through Essbase clients.

Drill-through URL XML

The drill-through URL XML is a block of XML information structured in a protocol that enables Essbase to link specified database regions to information on Oracle ERP and EPM applications. This URL XML is transparent to the end users querying the application. The URL XML is populated by the ERP or EPM applications that deployed the Essbase database. It is not recommended that the Administrator edit the URL XML; however, Essbase does provide the interface to edit the URL XML. The XML block contains the drill-through URL display name, as well as a URL enabling the hyperlink from a cell to a Web interface to occur.

List of Drillable Regions

The list of drillable regions is a member specification defining areas of the database that should allow drill-through using the specified URL. The administrator defines the list of drillable regions using a member specification of members from one or more dimensions. Define the member specification using the same Essbase member-set calculation language that you use for defining security filters. For example, the following is a valid member specification, indicating all eastern

states, except New York, for months of Qtr1: @REMOVE(@DESCENDANTS("Eastern Region"), “New York”), @CHILDREN(Qtr1).

Level-0 Boolean Flag

This flag indicates whether the URL applies only to level-0 descendents of the region specified by the list of drillable regions.

For example, if the level-0 flag is enabled for drillable region DESCENDANTS("Market"), @CHILDREN(Qtr1), then the URL is applicable for all states of Market during all months of Qtr1, and for all level-0 members across remaining dimensions.

Creating and Managing Drill-through URLs

Use the following MaxL statements to manage drill-through URLs:

- create drillthrough
- alter drillthrough
- drop drillthrough
- display drillthrough

See “MaxL Statements” on page 5.

Use the following Essbase API structures and functions to manage the drill-through URLs on the Essbase outline:

- C Main API Structure:
  - ESS_DURLINFO_T

  See “C Main API Structure” on page 10.
- C Main API functions:
  - EssCreateDrillThruURL
  - EssDeleteDrillThruURL
  - EssGetDrillThruURL
  - EssGetCellDrillThruReports
  - EssListDrillThruURLs
  - EssMDXIsCellGLDrillable
  - EssUpdateDrillThruURL

  See “C Main API Functions” on page 10.
- C Grid API structure:
  - ESSG_DATA_T

  See “C Grid API Structure” on page 19.
- C Grid API function:

- EssGGetIsCellDrillable

See "C Grid API Function" on page 20.

- Visual Basic API structure:
  - ESB_DURLINFO_T

See "Visual Basic API Structure" on page 21.

- Visual Basic API functions:
  - EsbCreateDrillThruURL
  - EsbUpdateDrillThruURL
  - EsbDeleteDrillThruURL
  - EsbListDrillThruURLs
  - EsbGetDrillThruURL
  - EsbGetCellDrillThruReports

See "Visual Basic API Functions" on page 22.

Use the following Oracle Essbase Administration Services topics to manage drill-through URLs:

- "Managing Drill-through Definitions" on page 50
- "Edit Drill-Through Definitions Dialog Box" on page 51

MaxL Statements

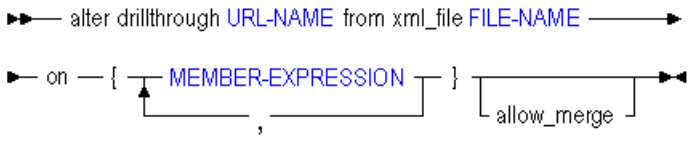
The following MaxL statements are for the drill-through to Oracle applications feature.

- `Alter Drillthrough`
- `Create Drillthrough`
- `Display Drillthrough`
- `Drop Drillthrough`

Alter Drillthrough

Edit drill-through URL definitions used to link to content hosted on Oracle ERP and EPM applications.

Syntax



Use **alter drillthrough** in the following ways to edit a URL definition.

Keyword	Description

alter drillthrough	Edit drill-through URL metadata.
from xml_file	Indicate the path to the local URL XML file that defines the link information. The URL XML is created by the ERP or EPM application that deployed the Essbase database. The XML contains the drill-through URL display name and a URL enabling the hyperlink from a cell to a Web interface to occur. For a sample URL XML file, see `Create Drillthrough`.
on {<member-expression>,...}	Define the list of drillable regions, using the same Essbase member-set calculation language that is used to define security filters. The list of drillable regions must be enclosed in {brackets}.
allow_merge	**Optional**: Merge the drillable-region definition instead of replacing it on update.

Example

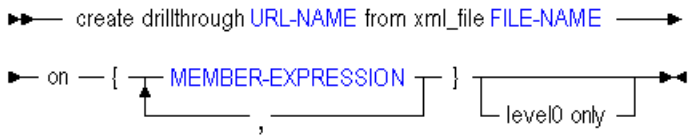
```
alter drillthrough sample.basic.myURL from xml_file "C:/drillthrough/data/
myfile.xml" on {'@Ichildren("Qtr1")', '@Ichildren("Qtr2")'} allow_merge;
```

Create Drillthrough

Create a drill-through URL within the active database outline.

For each drillable region of an Essbase database, you can enable drill-through access by means of a URL to Web content hosted on Oracle ERP and EPM applications.

Syntax



Use **create drillthrough** to create a drill-through URL definition in the following ways:

Keyword	Description
create drillthrough	Create a drill-through URL as metadata.

from xml_file	Indicate the path to the local URL XML file that defines the link information.
	The URL XML is created by the ERP or EPM application that deployed the Essbase database. The XML contains the drill-through URL display name and a URL enabling the hyperlink from a cell to a Web interface to occur.
	The following is a sample URL XML file:
	`<?xml version="1.0" encoding="UTF-8"?>`
	`<foldercontents path="/">`
	`  <resource name="Assets Drill through Fusion GL"`
	`description="" type="application/x-hyperion-`
	`applicationbuilder-report">`
	`  <name xml:lang="fr">Rapport de ventes</name>`
	`  <name xml:lang="es">Informe de ventas</name>`
	`  <action name="Display HTML" description="Launch`
	`HTML display of Content" shortdesc="HTML">`

```
      <url>/fusionapp/Assetsdrill.jsp?$SSO_TOKEN$&
$CONTEXT$&$ATTR(ds,pos,gen,level.edge)$
      </url>
    </action>
  </resource>
</foldercontents>
```

on {<member-expression>,...}	Define the list of drillable regions, using the same Essbase member-set calculation language that is used to define security filters. The list of drillable regions must be enclosed in {brackets}.
level0 only	**Optional**: Restrict the URL definition to level-0 data.

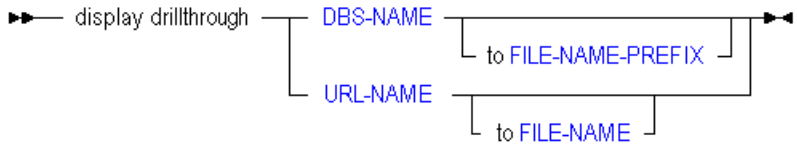
Example

```
create drillthrough sample.basic.myURL from xml_file "C:/drillthrough/data/
myfile1.xml" on {'@Ichildren("Qtr1")', '@Ichildren("Qtr2")'} level0 only;
```

Display Drillthrough

View drill-through URL definitions used to link to content hosted on Oracle ERP and EPM applications.

Syntax



You can display URL information in the following ways using **display user**.

Keyword	Description
<dbs-name>	Display all drill-through URL definitions on the database.
<dbs-name> to <file-name-prefix>	Display all drill-through URL definitions on the database, writing the URL XML content to file names prefixed with the string given as input for FILE-NAME-PREFIX.
<url-name>	Display the specified drill-through URL definition.
<url-name> to <file-name>	Display the specified drill-through URL definition, writing the URL XML content to the specified file name.

Example

```
display drillthrough sample.basic;
```

Displays all drill-through URL definitions on Sample.Basic.

```
display drillthrough sample.basic to "urlxmls";
```

Displays all drill-through URL definitions on Sample Basic, writing the URL XML content to file names prefixed with `urlxmls`.

```
display drillthrough sample.basic."Drill through To EPMI";
```

Displays the drill-through URL definition named `Drill through To EPMI`.

```
display drillthrough sample.basic."Drill through To EPMI" to "c:/temp/
drillthrough.xml";
```

Displays the drill-through URL definition named `Drill through To EPMI`, writing the URL XML content to the file `drillthrough.xml`.

Drop Drillthrough

Delete a drill-through URL definition used to link to content hosted on Oracle ERP and EPM applications.

Syntax

Example

```
drop drillthrough sample.basic.myURL;
```

MaxL Definitions

The following MaxL definitions are for the drill-through to Oracle applications feature.

- "FILE-NAME-PREFIX" on page 8
- "MEMBER-EXPRESSION" on page 9
- "URL-NAME" on page 9

FILE-NAME-PREFIX

Prefix for one or more file names to be created (upon **display drillthrough DBS-NAME to FILE-NAME-PREFIX**) on the client in the working directory of MaxL execution.

These display output files contain the URL XML content of URL drill-through definitions used to link to content hosted on ERP and EPM applications.

If the string contains special characters, it must be enclosed in single or double quotation marks.

Type

string

Example

```
urlxmls
```

Referenced By

`Display Drillthrough`

MEMBER-EXPRESSION

Outline member specification of members from one or more dimensions, member combinations separated by commas, or member sets defined with functions. Must be enclosed in single or double quotation marks.

Type

string

Example

```
'@ANCESTORS(Qtr2)'
```

If MEMBER-EXPRESSION contains MEMBER-NAMES that begin with numbers or contain special characters, then enclose those member names in double quotation marks, and the entire MEMBER EXPRESSION in single quotation marks. For example:

- `create or replace filter demo.basic.numfilt no_access on '"2"';`
- `'@DESCENDANTS("Eastern Region"), @CHILDREN(Qtr1)'`

The following example shows how `Create Drillthrough` uses a member expression to define the list of drillable regions.

```
create drillthrough sample.basic.myURL from xml_file "temp.xml" on
{'@Ichildren("Qtr1")', '@Ichildren("Qtr2")'} level0 only;
```

Referenced By

`Alter Drillthrough`

`Create Drillthrough`

URL-NAME

The name of a drill-through URL definition used to link to content hosted on Oracle ERP and EPM applications.

Syntax

name1.name2.name3

- *name1*—Application name
- *name2*—Database name
- *name3*—URL name

Type

name

Example

```
Sample.basic.MyURL
```

If any part of the name contains special characters, the name must be enclosed in single or double quotation marks.

Referenced By

Alter Drillthrough

Create Drillthrough

Display Drillthrough

Drop Drillthrough

C Main API Structure

The following C Main API structure is for the drill-through to Oracle applications feature.

ESS_DURLINFO_T

ESS_DURLINFO_T is a data structure for capturing drill-through URL information.

```
typedef struct url
{
   ESS_CHAR_T     bIsLevel0;
   ESS_STR_T      cpURLName;
   ESS_USHORT_T   iURLXmlSize;
   ESS_BYTE_T*    cpURLXml;
   ESS_USHORT_T   iCountOfDrillRegions;
   ESS_PSTR_T     cppDrillRegions;
}  ESS_DURLINFO_T;
```

Data Type	Field	Description
ESS_STR_T	*cpURLName*	Name of the drill-through URL
ESS_CHAR_T	*bIsLevel0*	If 1, then URL definition is restricted to level-0 data; if 0, there is no restriction
ESS_USHORT_T	*iURLXmlSize*	Size of the URL XML text
ESS_BYTE_T*	*cpURLXml*	Pointer to the URL XML text
ESS_USHORT_T	*icountOfDrillRegions*	Number of regions referenced by the drill-through URL
ESS_PSTR_T	*cppdrillRegions*	List of regions referenced by the drill-through URL

C Main API Functions

The following C Main API functions are for the drill-through to Oracle applications feature.

- EssCreateDrillThruURL
- EssDeleteDrillThruURL
- EssGetCellDrillThruReports
- EssGetDrillThruURL
- EssListDrillThruURLs
- EssMDXIsCellGLDrillable
- EssUpdateDrillThruURL

EssCreateDrillThruURL

Creates a drill-through URL, with the given link and name, within the active database outline.

Syntax

```
ESS_FUNC_M EssCreateDrillThruURL (hCtx, pUrl);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
pUrl	ESS_PDURLINFO_T	URL definition

Return Value

- If successful, creates a drill-through URL in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

```
/* Sample Code for EssCreateDrillThruURL */

ESS_STS_T sts = ESS_STS_NOERR;
ESS_DURLINFO_T url;
ESS_USHORT_T usCountOfURLs, i;
ESS_PDURLINFO_T listOfURLs;
ESS_STR_T urlName = "";
ESS_PDURLINFO_T urlInfo;
ESS_STR_T fileName = "";
ESS_CHAR_T xmlString[XML_CHAR_MAX];

/* Valid case */

memset(&url, '\0', sizeof(ESS_DURLINFO_T));
fileName = "F:\\testarea\\mainapi\\sample1.xml";
GetFileContent(fileName, xmlString);
```

```
printf("\nValid case:\n");
   url.bIsLevel0 = ESS_TRUE;
   url.cpURLName = "Drill Through to EPMI";
   url.cpURLXml = xmlString;
   url.iURLXmlSize = (ESS_SHORT_T) strlen(xmlString)+1;
   url.iCountOfDrillRegions = 2;
sts = EssAlloc (hInst, sizeof(ESS_STR_T) * url.iCountOfDrillRegions,
&(url.cppDrillRegions));
   url.cppDrillRegions[0] = "@idesc(\"Qtr1\")";
   url.cppDrillRegions[1] = "@idesc(\"Qtr2\")";
sts = EssCreateDrillThruURL(hCtx, &url);
printf("EssCreateDrillThruURL sts: %ld\n",sts);
```

EssDeleteDrillThruURL

Deletes a drill-through URL, with the given URL name, within the active database outline.

Syntax

```
ESS_FUNC_M EssDeleteDrillThruURL (hCtx, URLName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
URLName	ESS_STR_T	Drill-through URL name

Return Value

- If successful, deletes the named drill-through URL in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

```
ESS_STS_T sts = ESS_STS_NOERR;

sts = EssDeleteDrillThruURL(hCtx, "Drill Through to EPMI");
printf("EssDeleteDrillThruURL sts: %ld\n",sts);
```

EssGetCellDrillThruReports

Gets the drill-through reports associated with a data cell as a list of URL XMLs, given the cell's member combination.

Syntax

```
ESS_FUNC_M EssGetCellDrillThruReports (hCtx, noMbrs, pMbrs, nURLXML,
ppURLXMLLen, ppURLXML);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
noMbrs	ESS_USHORT_T	Number of members in the member list *pMbrs*
pMbrs	ESS_PSTR_T	Pointer to the list of member names (or Aliases); the array size is assumed to be the dimension count
nURLXML	ESS_PUSHORT_T	Number of URL XMLs returned
ppURLXMLLen	ESS_PPUSHORT_T	Returns length of URL XML generated
ppURLXML	ESS_PPVOID_T	Returns pointers to the URL XML byte stream

Notes

The application database must be set to Active for this call. This function must be extended to support any additional information needed by the clients.

Return Value

- If successful, gets the list of URL XMLs.
- If unsuccessful, returns an error code.

Access

- Caller must have database Read privilege (ESS_PRIV_READ) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

```
/* Sample Code for EssGetCellDrillThruReports */

ESS_STS_T sts = ESS_STS_NOERR;
ESS_SHORT_T    numMbrs = 0;
ESS_STR_T    *pMbrs = ESS_NULL;
ESS_USHORT_T    numURLXML, i = 0;
ESS_USHORT_T    *URLXMLLen = ESS_NULL;
ESS_PPVOID_T    *URLXML = ESS_NULL;
ESS_CHAR_T    pTmpXML[XML_CHAR_MAX];

/* Valid case */

numMbrs = 5;
sts = EssAlloc (hInst, sizeof(ESS_STR_T) * numMbrs , &pMbrs);
pMbrs[0] = "Jul";
pMbrs[1] = "100-10";
pMbrs[2] = "Actual";
pMbrs[3] = "New York";
pMbrs[4] = "Sales";
sts = EssGetCellDrillThruReports(hCtx, numMbrs, pMbrs, &numURLXML,
&URLXMLLen, &URLXML);
printf("EssGetCellDrillThruReports sts: %ld\n",sts);
   if(!sts)
   {
      printf("\nNumber of URL XML: %d", numURLXML);
```

```
      for (i = 0; i < numURLXML; i++)
      {
         memset(pTmpXML, 0, XML_CHAR_MAX);
         memcpy(pTmpXML, URLXML[i], URLXMLLen[i]);

         if ( URLXML[i] != ESS_NULL )
            printf("\tXML [%d] : %s\n", i, pTmpXML );
         else
            printf("\tXML [%d] : NULL STRING \n", i );
      if ( URLXML[i] != ESS_NULL )
         EssFree(hInst, URLXML[i]);
      }
   if ( URLXML != ESS_NULL )
      EssFree(hInst, URLXML);
   if ( URLXMLLen != ESS_NULL )
      EssFree(hInst, URLXMLLen);
   }
```

EssGetDrillThruURL

Gets the drill-through URL within the active database outline.

Syntax

```
ESS_FUNC_M EssGetDrillThruURL (hCtx, URLName, &pUrl);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
URLName	ESS_STR_T	Drill-through URL name
pUrl	ESS_PDURLINFO_T	URL definition

Return Value

- If successful, gets the drill-through URL in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Read privilege (ESS_PRIV_READ) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

```
static void DisplayUrlDefn (ESS_PDURLINFO_T pUrls )
{
   ESS_UINT_T    i;

   printf("\tUrlname         : %s\n", pUrls->cpURLName );
   if (pUrls->bIsLevel0)
      printf("\tUrl Is Level-0 slice : Yes\n");
   else
      printf("\tUrl Is Level-0 slice : No\n");
```

```
   printf("\tUrlXmlsize    : %i\n", pUrls->iURLXmlSize );
   printf("\tUrlXml        : %s\n", (ESS_STR_T) pUrls->cpURLXml);

   printf("\tNumber of drill region(s): %d\n", pUrls-
>iCountOfDrillRegions);
   for ( i = 0; i < pUrls->iCountOfDrillRegions; i++ )
   {
   printf("\t\tDrillRegion[%d]: %s\n", i, pUrls->cppDrillRegions[i] );
   }
   printf("\n");
}
ESS_STS_T sts = ESS_STS_NOERR;
ESS_STR_T urlName = "";
ESS_USHORT_T usCountOfURLs, i;
ESS_PDURLINFO_T urlInfo;

/* Valid case*/

urlName = "Drill Through to EPMI";
sts = EssGetDrillThruURL(hCtx, urlName, &urlinfo);
printf("EssGetDrillThruURL sts: %ld\n",sts);
if(!sts)
   DisplayUrlDefn(urlInfo);

EssFreeStructure (hInst, ESS_DT_STRUCT_URLINFO, 1, (ESS_PVOID_T)urlInfo);
```

EssListDrillThruURLs

Lists the drill-through URL names within the active database outline.

Syntax

```
ESS_FUNC_M EssListDrillThruURLs (hCtx, &pCountOfUrls, &pUrls);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
pCountOfUrls	ESS_PUSHORT_T	Count of drill-through URLs
pUrls	ESS_PPDURLINFO_T	List of URLs

Notes

The ESS_DURLINFO_T structure array must be deallocated by the caller using EssFreeStructure() with the ESS_DT_STRUCT_URLINFO option.

Return Value

- If successful, lists drill-through URLs in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Read privilege (ESS_PRIV_READ) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

```
static void DisplayUrlDefn (ESS_PDURLINFO_T pUrls )
{
   ESS_UINT_T    i;

   printf("\tUrlname        : %s\n", pUrls->cpURLName );
   if (pUrls->bIsLevel0)
      printf("\tUrl Is Level-0 slice : Yes\n");
   else
      printf("\tUrl Is Level-0 slice : No\n");

   printf("\tUrlXmlsize    : %i\n", pUrls->iURLXmlSize );
   printf("\tUrlXml       : %s\n", (ESS_STR_T) pUrls->cpURLXml);

   printf("\tNumber of drill region(s): %d\n", pUrls-
>iCountOfDrillRegions);
   for ( i = 0; i < pUrls->iCountOfDrillRegions; i++ )
   {
      printf("\t\tDrillRegion[%d]: %s\n", i, pUrls->cppDrillRegions[i] );
   }
   printf("\n");
}

ESS_STS_T sts = ESS_STS_NOERR;
ESS_USHORT_T usCountOfURLs, i;
ESS_PDURLINFO_T listOfURLs;
ESS_DURLINFO_T url;

/* Valid case*/

sts = EssListDrillThruURLs(hCtx, &usCountOfURLs, &listOfURLs);
printf("EssListDrillThruURLs sts: %ld\n",sts);
if(!sts)
   {
      printf("\tCount of URL: %d\n", usCountOfURLs);
      printf("\tList of URL(s):\n");
      for(i = 0; i < usCountOfURLs; i++)
      {
         DisplayUrlDefn (&listOfURLs[i]);
      }
   }
EssFreeStructure (hInst, ESS_DT_STRUCT_URLINFO, usCountOfURLs, listOfURLs);
```

EssMDXIsCellGLDrillable

Checks whether the cell is associated with a drill-through URL.

Syntax

```
ESS_FUNC_M EssMdxIsCellGLDrillable (hQry, hCell, pIsDrillable);
```

Parameter	Data Type	Description
hQry	ESS_MDX_QRYHDL_T	Query handle
hCell	ESS_MDX_CELLHDL_T	Cell handle

pIsDrillable	ESS_PBOOL_T	True, if the cell is associated with a drill-through URL; False, otherwise

Return Value

- If successful, sets *pIsDrillable* based on the cell's status.
- If unsuccessful, returns an error message.

Example

```
#define ESS_MDX_CELLPROP_GLDRILLTHRU          0x00000008

   if ((sts = EssMdxNewQuery(hCtx, qry, &hQry)) != ESS_STS_NOERR)
   {
      printf("EssMdxNewQuery failure: %ld\n", sts);
      exit ((int) sts);
   }
   printf("EssMdxNewQuery sts: %ld\n", sts);

   if ((sts = EssMdxSetQueryCellProperties(hQry,
          (ESS_MDX_CELLPROP_GLDRILLTHRU
          )
      )) != ESS_STS_NOERR)
   {
      printf("EssMdxSetQueryCellProperties failure: %ld\n", sts);
      exit ((int) sts);
   }
   if ((sts = EssMdxExecuteQuery(hQry)) != ESS_STS_NOERR)
   {
      printf("EssMdxExecuteQuery failure: %ld\n", sts);
      exit ((int) sts);
   }
   printf("EssMdxExecuteQuery sts: %ld\n", sts);

/* To retrieve IsCellGLDrillable property of a cell, use
EssMdxIsCellGLDrillable*/

   if ((sts = EssMdxIsCellGLDrillable(hQry, hCell, &bIsCellGLDT))
      != ESS_STS_NOERR)
   {
      printf("EssMdxIsCellGLDrillable failure: %ld\n", sts);
         exit ((int) sts);
   }
   if (bIsCellGLDT)
      printf(" Is Cell Drillable: TRUE\n");
   else
      printf(" Is Cell Drillable: FALSE\n");
```

EssUpdateDrillThruURL

Updates a drill-through URL, with the given name, within the active database outline.

Syntax

```
ESS_FUNC_M EssUpdateDrillThruURL (hCtx, ESS_PDURLINFO_T pUrl);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
pUrl	ESS_PDURLINFO_T	URL definition
bMerge	ESS_BOOL_T	• If True, add drill-through region definitions in *pUrl* to the existing list of drill-through regions in the named URL definition • If False, replace the existing list of drill-through region definitions with the list in *pUrl*

Return Value

- If successful, updates the named drill-through URL in the active database by replacing the URL XML and either updating or replacing the drill-through region list with the corresponding fields in *pUrl*.
- If there is no URL with the given name, returns an error code.

Access

- Caller must have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

```
/* Sample Code for EssUpdateDrillThruURL */

ESS_STS_T sts = ESS_STS_NOERR;
ESS_DURLINFO_T url;
ESS_PDURLINFO_T urlInfo;
ESS_STR_T fileName = "";
ESS_CHAR_T xmlString[XML_CHAR_MAX];
ESS_BOOL_T bMerge;
ESS_USHORT_T i;

memset(&url, '\0', sizeof(ESS_DURLINFO_T));
fileName = "F:\\testarea\\mainapi\\sample1.xml";
GetFileContent(fileName, xmlString);

/* Update URL*/
url.bIsLevel0 = ESS_TRUE;
url.cpURLName = "Drill Through to EPMI";
url.cpURLXml = xmlString;
url.iURLXmlSize = (ESS_SHORT_T) strlen(xmlString)+1;
url.iCountOfDrillRegions = 1;
sts = EssAlloc (hInst, sizeof(ESS_STR_T) * url.iCountOfDrillRegions,
&(url.cppDrillRegions));

/* With bMerge = ESS_FALSE, update Drill Regions */

bMerge = ESS_FALSE;   // replace
url.cppDrillRegions[0] = "Mar";
sts = EssUpdateDrillThruURL(hCtx, &url, bMerge);
printf("EssUpdateDrillThruURL sts: %ld\n",sts);
```

C Grid API Structure

The following C Grid API structure is for the drill-through to Oracle applications feature.

ESSG_DATA_T

Describes the format of the data to be sent and received by the Essbase Grid API. Note that calls returning this structure will return member names in the Member structure. The caller can pass in the same structure back to the API using the Member structure instead of the *pszStr* field if the type is ESSG_DT_MEMBER.

The ESSG_DATA_T data structure defines each cell sent or returned via the grid API. If this structure is being returned to the caller, *pszStr* contains string data and *dblData* contains numeric data. Use the *usType* field to determine whether the cell is a member, a number, or text. Similarly, if the structure is being passed into the API, *pszStr* should contain a member name or text and *dblData* should contain numeric data. Set the *usType* field to correspond to the data type of the cell. If the cell data type is unknown, set it to text (ESSG_DT_STRING), and the server determines whether it is a member.

```
typedef struct ESSG_DATA_T
{
   ESSG_PVOID_T    pAttributes;
   ESSG_DATA_VALUE Value;
   ESSG_USHORT_T   usType;
   ESSG_PVOID_T    pCellProps;
} ESSG_DATA_T;

ESS_TSA_API_typedef(ESSG_DATA_T *,  ESSG_PDATA_T);
ESS_TSA_API_typedef(ESSG_DATA_T **, ESSG_PPDATA_T);
```

Data Type	Field	Description
ESSG_PVOID_T	*pAttributes*	One of the long integer constants listed below indicating the cell type or member type (OUT)
ESSG_DATA_VALUE_T	*Value*	The value of the returned grid string
ESSG_USHORT_T	*usType*	One of the tag constants listed below indicating the data type (IN/OUT)
ESSG_PVOID_T	*pCellProps*	Stores cell properties; for example, whether or not cell is associated with a drill-through URL

Constants for ESSG_DATA_T

The following constants are used by the *pAttributes* field of the ESSG_DATA_T structure for cell data types:

```
ESSG_CA_READONLY
ESSG_CA_READWRITE
ESSG_CA_LINKEDOBJ
ESSG_CA_LINKPARTITION
ESSG_CA_LINKCELLNOTE
```

```
ESSG_CA_LINKWINAPP
ESSG_CA_LINKURL
ESSG_CA_AISDT
ESSG_CA_GLDT
```

The following constants are used by the *pAttributes* field of the ESSG_DATA_T structure for member data types:

```
ESSG_MA_DIMTOP
ESSG_MA_ZOOMINABLE
ESSG_MA_NEVERSHARE
ESSG_MA_LABELONLY
ESSG_MA_STOREDATA
ESSG_MA_EXPSHARE
ESSG_MA_IMPSHARE
ESSG_MA_DYNCALC
ESSG_MA_FORMULA
ESSG_MA_ATTRIBUTE
ESSG_MA_DIMNUMBITS
```

The following constants are used by the *usType* field of the ESSG_DATA_T structure:

```
ESSG_DT_UNUSED
ESSG_DT_STRING
ESSG_DT_LONG
ESSG_DT_DOUBLE
ESSG_DT_BLANK
ESSG_DT_RESERVED
ESSG_DT_ERROR
ESSG_DT_MISSING
ESSG_DT_ZERO
ESSG_DT_NOACCESS
ESSG_DT_MEMBER
ESSG_DT_FORMULA
ESSG_DT_ZEROwFORMULA
ESSG_DT_DOUBLEwFORMULA
ESSG_DT_BLANKwFORMULA
ESSG_DT_STRINGwFORMULA
ESSG_DT_MISSINGwFORMULA
ESSG_DT_NOACCESSwFORMULA
ESSG_DT_STRINGEX
ESSG_DT_MEMBEREX
ESSG_DT_STRINGEXwFORMULA
ESSG_DT_FORMULAEX
ESSG_DT_MEMBERwKEY
```

C Grid API Function

The following C Grid API function is for the drill-through to Oracle applications feature.

EssGGetIsCellDrillable

Checks whether a cell is associated with a drill-through URL.

Syntax

```
ESS_FUNC_M EssGGetIsCellDrillable (hGrid, pData, pIsDrillable);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Grid handle returned by **EssGNewGrid**()
pData	ESS_PDATA_T	Pointer to the ESSG_DATA_T structure of the cell
pIsDrillable	ESS_PBOOL_T	True, if the cell is associated with a drill-through URL; False otherwise

Return Value

- If successful, sets *pIsDrillable* accordingly.
- If unsuccessful, returns an error code.

Example

```
#define   ESSG_OP_GET_DRILLTHRU_URLS            41

ESSG_STS_T  sts = EssGInit(&InitStruct, &Handle);
sts = EssGNewGrid(Handle, &hGrid);
sts =
EssGConnect(hGrid,Server,UserName,Password,Application,Database,ulOptions);
sts = EssGSetGridOption(hGrid, ESSG_OP_GET_DRILLTHRU_URLS ,(ESSG_PVOID_T)
(ESSG_TRUE));

ppDataIn = BuildQuery(&rRangeDataIn);

sts = EssGBeginRetrieve(hGrid,ESSG_RET_RETRIEVE);
sts = EssGSendRows(hGrid, &rRangeDataIn, ppDataIn);
sts = EssGPerformOperation(hGrid, 0);

/*To retrieve the cell drillable property of a cell*/

EssGGetIsCellDrillable(hGrid, &(cells[ulRow][ulCol]), &bIsDrillable);
   if (bIsDrillable)
      printf("bIsDrillable: true");
   else
      printf("bIsDrillable: false");
```

Visual Basic API Structure

The following Visual Basic API structure and functions are for the drill-through to Oracle applications feature.

ESB_DURLINFO_T

A data structure used to capture URL information. The fields are:

```
Type ESB_DURLINFO_T
   bIsLevel0    As Integer  'consider level-0 members along symmetric
regions
   iURLXMLSize  As Integer        'URL XML size
   cpURLName    As String * 1024  'URL identifier
```

```
   cpURLXML     As String * 8192  'URL XML
End Type
```

Visual Basic Data Type	Field	Description
As Integer	*bIsLevel0*	If 1, then URL definition is restricted to level-0 data; if 0, there is no restriction
As Integer	*iURLXMLSize*	Size of URL XML
As String * 1024	*cpURLName*	Name of URL definition
As String * 8192	*cpURLXML*	Content of URL XML

Note: The regions list is passed as a separate argument, *symRegions()*, within each Visual Basic drill-through function.

Visual Basic API Functions

The following Visual Basic API functions are for the drill-through to Oracle applications feature.

- EsbCreateDrillThruURL
- EsbDeleteDrillThruURL
- EsbGetCellDrillThruReports
- EsbGetDrillThruURL
- EsbListDrillThruURLs
- EsbUpdateDrillThruURL

EsbCreateDrillThruURL

Creates a drill-through URL, with the given link and the name, within the active database outline.

Syntax

```
Declare Function EsbCreateDrillThruURL Lib "esbapin" (ByVal hCtx As Long,
ByRef symRegions() As String, ByRef pUrl As ESB_DURLINFO_T) As Long
```

Parameter	Description
hCtx	Visual Basic API context handle
symRegions()	Array containing the symmetric region specification
pUrl	URL definition

Return Value

- If successful, creates a drill-through URL in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as their active database using EsbSetActive().

Example

```
Sub ESB_CreateGLDrillThru()
   Dim sts                         As Long
   Dim url                         As ESB_DURLINFO_T
   Dim cppDrillRegions(0 To 1)     As String

   '*************************************************************
   ' Need to create a local context, if files are not on the server
   '*************************************************************
   url.bIsLevel0 = 0

   cppDrillRegions(0) = "sales"
   cppDrillRegions(1) = "cogs"
   url.cpURLXML = "Testing"
   url.cpURLName = "VB URL7"
   url.iURLXMLSize = 8

   sts = EsbCreateDrillThruURL(hCtx, cppDrillRegions, url)

   Debug.Print "EsbCreateDrillThruURL sts: " & sts
End Sub
```

See also an extended example in "Drill-through Visual Basic API Example" on page 28.

EsbDeleteDrillThruURL

Deletes a drill-through URL, with the given URL name, within the active database outline.

Syntax

```
Declare Function EsbDeleteDrillThruURL Lib "esbapin" (ByVal hCtx As Long,
ByVal URLName As String) As Long
```

Parameter	Description
hCtx	Visual Basic API context handle
URLName	Drill-through URL name

Return Value

- If successful, deletes the named drill-through URL in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

- Caller must have selected the specified database as their active database using EsbSetActive().

Example

```
Sub ESB_DeleteGLDrillThru()
   Dim URLName      As String

   URLName = "VB URL7"
   sts = EsbDeleteDrillThruURL(hCtx, URLName)

  Debug.Print "EsbDeleteDrillThruURL sts: " & sts
End Sub
```

See also an extended example in "Drill-through Visual Basic API Example" on page 28.

EsbGetCellDrillThruReports

Gets the drill-through reports associated with a data cell as a list of URL XMLs, given the cell's member combination.

Syntax

```
Declare Function EsbGetCellDrillThruReports Lib "esbapin" (ByVal hCtx As
Long, ByRef pMbrs() As String, ByRef ppURLXMLLen As Variant, ByRef ppURLXML
As Variant) As Long
```

Parameter	Description
hCtx	Visual Basic API context handle
pMbrs	List of member names (or Aliases)
ppURLXMLLen	Returns length of URL XML generated
ppURLXML	Returns pointers to the URL XML byte stream

Notes

The application database needs to be set to Active for this call. This function needs to be extended to support any additional information needed by the clients.

Return Value

- If successful, gets the list of URL XMLs.
- If unsuccessful, returns an error code.

Access

- Caller must have database Read privilege (ESB_PRIV_READ) for the specified database.
- Caller must have selected the specified database as their active database using EsbSetActive().

Example

```
Sub ESB_GetCellDrillThruReports()
   Dim intX           As Integer
   Dim mbrs(0 To 4)   As String
   Dim pURLXMLLens    As Variant
```

```
    Dim pURLXMLs       As Variant

    mbrs(0) = "sales"
    mbrs(1) = "jan"
    mbrs(2) = "New York"
    mbrs(3) = "actual"
    mbrs(4) = "100-10"

    sts = EsbGetCellDrillThruReports(hCtx, mbrs, pURLXMLLens, pURLXMLs)

    If sts = 0 Then

        Debug.Print "EsbGetCellDrillThruReports sts: " & sts
        For intX = LBound(pURLXMLLens) To UBound(pURLXMLLens)

        Debug.Print "URL XML: " & intX
        Debug.Print "URL XML Len: " & pURLXMLLens(intX)
        Debug.Print "URL XML String: " & pURLXMLs(intX)

        Next
    End If

    mbrs(0) = "profit"
    sts = EsbGetCellDrillThruReports(hCtx, mbrs, pURLXMLLens, pURLXMLs)
    If sts = 0 Then
        Debug.Print "EsbGetCellDrillThruReports sts: " & sts
        For intX = LBound(pURLXMLLens) To UBound(pURLXMLLens)
        Debug.Print "URL XML: " & intX
        Debug.Print "URL XML Len: " & pURLXMLLens(intX)
        Debug.Print "URL XML String: " & pURLXMLs(intX)
        Next
    End If
End Sub
```

See also an extended example in "Drill-through Visual Basic API Example" on page 28.

EsbGetDrillThruURL

Gets a list of drill-through URL names within the active database outline.

Syntax

```
Declare Function EsbGetDrillThruURL Lib "esbapin" (ByVal hCtx As Long,
ByVal URLName As String, pUrl As ESB_DURLINFO_T, ByRef symRegions As
Variant) As Long
```

Parameter	Description
hCtx	Visual Basic API context handle
URLName	Drill-through URL name
pUrl	URL definition
symRegions	List of symmetric regions

Return Value

- If successful, gets a list of drill-through URLs in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Read privilege (ESB_PRIV_READ) for the specified database.
- Caller must have selected the specified database as their active database using EsbSetActive().

Example

```
Sub ESB_GetGLDrillThru()
   Dim URLName           As String
   Dim url               As ESB_DURLINFO_T
   Dim intX              As Integer
   Dim cppDrillRegions   As Variant

   URLName = "VB URL2"
   sts = EsbGetDrillThruURL(hCtx, URLName, url, cppDrillRegions)

   Debug.Print "EsbGetDrillThruURL sts: " & sts

   If sts = 0 Then
     Debug.Print "URL Name: " & url.cpURLName
     Debug.Print "URL XML: " & url.cpURLXML

     For intX = LBound(cppDrillRegions) To UBound(cppDrillRegions)

     Debug.Print "URL Region: " & cppDrillRegions(intX)

   Next
   End If
End Sub
```

See also an extended example in "Drill-through Visual Basic API Example" on page 28.

EsbListDrillThruURLs

Lists the drill-through URLs within the active database outline.

Syntax

```
Declare Function EsbListDrillThruURLs Lib "esbapin" (ByVal hCtx As Long,
ByRef URLNames As Variant) As Long
```

Parameter	Description
hCtx	Visual Basic API context handle
URLNames	List of URL names

Return Value

- If successful, lists names of drill-through URLs in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Read privilege (ESB_PRIV_READ) for the specified database.
- Caller must have selected the specified database as their active database using EsbSetActive().

Example

```
Sub ESB_ListGLDrillThru()
   Dim intX       As Integer
   Dim URLNames   As Variant

   sts = EsbListDrillThruURLs(hCtx, URLNames)

   If sts = 0 Then
      Debug.Print "EsbListDrillThruURLs sts: " & sts

      For intX = LBound(URLNames) To UBound(URLNames)

      Debug.Print "URL Name: " & URLNames(intX)

      Next
   End If
End Sub
```

See also an extended example in "Drill-through Visual Basic API Example" on page 28.

EsbUpdateDrillThruURL

Updates a drill-through URL, with the given name, within the active database outline.

Syntax

```
Declare Function EsbUpdateDrillThruURL Lib "esbapin" (ByVal hCtx As Long,
ByRef symRegions() As String, ByRef pUrl As ESB_DURLINFO_T, ByVal bMerge As
Integer) As Long
```

Parameter	Description
hCtx	Visual Basic API context handle
symRegions()	Array containing the symmetric region specification
pUrl	URL definition
bMerge	• If True, add drill-through region definitions in *pUrl* to the existing list of drill-through regions in the named URL definition • If False, replace the existing list of drill-through region definitions with the list in *pUrl*

Return Value

- If successful, updates the named drill-through URL in the active database by replacing the URL XML and either updating or replacing the drill-through region list with the corresponding fields in *pUrl*.
- If there is no URL with the given name, returns an error code.

Access

- Caller must have database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as their active database using EsbSetActive().

Example

```
Sub ESB_UpdateGLDrillThru()
   Dim sts                        As Long
   Dim url                        As ESB_DURLINFO_T
   Dim cppDrillRegions(0 To 1)    As String
   Dim bMerge                     As Integer

   '*************************************************************
   ' Need to create a local context, if files are not on the server
   '*************************************************************
   url.bIsLevel0 = 0
   bMerge = ESB_TRUE

   cppDrillRegions(0) = "qtr1"
   url.cpURLXML = "Testing"
   url.cpURLName = "VB URL7"
   url.iURLXMLSize = 8

   sts = EsbUpdateDrillThruURL(hCtx, cppDrillRegions, url, bMerge)

   Debug.Print "EsbUpdateDrillThruURL sts: " & sts
End Sub
```

See also an extended example in "Drill-through Visual Basic API Example" on page 28.

Drill-through Visual Basic API Example

```
Attribute VB_Name = "Module3"
Dim sts As Long
   Dim hInst As Long
   Dim hDestInst As Long
   Dim hCtx As Long
   Dim hDestCtx As Long
   Dim Server As String * ESB_SVRNAMELEN
   Dim User As String * ESB_USERNAMELEN
   Dim Password As String * ESB_PASSWORDLEN
   Dim AppName As String * ESB_APPNAMELEN
   Dim DbName As String * ESB_DBNAMELEN

Sub ESB_GetVersion()
   Dim sts As Long
   Dim Release As Integer
   Dim Version As Integer
   Dim Revision As Integer
   sts = EsbGetVersion(hCtx, Release, Version, Revision)
   Debug.Print "EsbGetVersion: sts = " & sts
   Debug.Print "Release: " & Release
   Debug.Print "Version: " & Version
   Debug.Print "Revision: " & Revision
```

```
End Sub

Sub ESB_Init()
   Dim Init As ESB_INIT_T

   ESB_FALSE = 0
   ESB_TRUE = 1

   Init.Version = ESB_API_VERSION
   Init.MaxHandles = 10
   Init.LocalPath = "C:\install\zolahit\products\Essbase\EssbaseClient"
   ' Use default message file
   Init.MessageFile = ""
   ' Use EsbGetMessage to retrieve
   ' messages
   Init.ClientError = ESB_TRUE
   Init.ErrorStack = 100
   'Init.vbCallbackFuncAddress = GetProcAddress(AddressOf EsbErrorHandler)

   sts = EsbInit(Init, hInst)
   'MsgBox ("EsbInit = " & sts)
   Debug.Print "EsbInit: sts = " & sts

   'For copy objects between servers
   'sts = EsbInit(Init, hDestInst)
   'MsgBox ("EsbInit = " & sts)
   'Debug.Print "EsbInit: sts = " & sts
End Sub

Public Function GetProcAddress(ByVal lngAddressOf As Long) As Long
   GetProcAddress = lngAddressOf
End Function

Public Function EsbErrorHandler(ByVal MsgNum As Long, ByVal Level As Long,
ByVal uLog As String, ByVal uMsg As String) As Long
   If Level >= ESB_LEVEL_ERROR Then
      MsgBox "Error: " & MsgNum & " - " & uMsg
   End If

   'MsgBox " Info " & MsgNum & ": Level: " & Level & ": " & uLog & ": " &
uMsg
End Function

Sub ESB_GetMessage()
   Dim DbName As String
   Dim FilterName As String
   Const szMessage = 256
   Dim Message As String * szMessage
   Dim Number As Long
   Dim Level As Integer
   Dim sts As Long
   Dim Object As ESB_OBJDEF_T
   Dim hOutline As Long
   Dim hMemberProfit As Long

   Object.hCtx = hCtx
   Object.Type = ESB_OBJTYPE_OUTLINE
```

```
   Object.AppName = "Temp"
   Object.DbName = "Basic"
   Object.FileName = "Basic"
   sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
   Debug.Print "EsbOtlOpenOutline: sts = " & sts

   sts = EsbOtlFindMember(hOutline, "100-10", hMember)
   Debug.Print "EsbOtlFindMember: sts = " & sts

   If sts > 0 Then
      sts = EsbGetMessage(hInst, Level, Number, Message, szMessage)
      Do While Mid$(Message, 1, 1) <> Chr$(0)
         Debug.Print Level
         Debug.Print Number
         Debug.Print Message
         sts = EsbGetMessage(hInst, Level, Number, Message, szMessage)
         Debug.Print "EsbGetMessage: sts = " & sts
       Loop
   End If
End Sub

Sub ESB_Login()
   Dim Items As Integer
   Dim AppDb As ESB_APPDB_T

   Server = "ppamu-pc1"
   User = "essexer"
   Password = "password"
   sts = EsbLogin(hInst, Server, User, Password, Items, hCtx)
   Debug.Print "EsbLogin: sts = " & sts
   'For n = 1 To Items
    ' sts = EsbGetNextItem(hCtx, ESB_LAPPDB_TYPE, AppDb)
    ' Debug.Print "EsbGetNextItem: sts = " & sts
    ' Debug.Print "App Name: "; AppDb.AppName
    ' Debug.Print "Db Name: "; AppDb.DbName
  ' Next

   'For copy objects between servers
   'sts = EsbLogin(hDestInst, "qtfsun1:1501", User, Password, Items,
hDestCtx)
   'Debug.Print "EsbLogin: sts = " & sts
End Sub

Sub ESB_AutoLogin()
   Dim pOption As Integer
   Dim pAccess As Integer

   Server = "localhost"
   'User = "essexer"
   'Password = "Password"
   'AppName = "sample"
   'DbName = "basic"

   'pOption = ESB_AUTO_NODIALOG + ESB_AUTO_NOSELECT
   pOption = ESB_AUTO_DEFAULT
   sts = EsbAutoLogin(hInst, Server, User, Password, AppName, DbName,
pOption, pAccess, hCtx)
```

```
   'MsgBox ("EsbAutoLogin = " & sts)
   Debug.Print "EsbAutoLogin: sts = " & sts
  ' Call Esb_runreport
End Sub

Sub ESB_LoginSetPassword()
   'Dim hInst       As Long
   'Dim Server      As String * ESB_SVRNAMELEN
   'Dim User        As String * ESB_USERNAMELEN
   'Dim Password    As String * ESB_PASSWORDLEN
   Dim NewPassword  As String * ESB_PASSWORDLEN
   Dim Items        As Integer
   Dim AppDb        As ESB_APPDB_T

   Server = "stiahp1:1501"
   User = "essexer"
   Password = "password"
   NewPassword = "password2"
   sts = EsbLoginSetPassword(hInst, Server, User, Password, NewPassword,
Items, hCtx)
   Debug.Print "EsbLoginSetPassword: sts = " & sts

   For N = 1 To Items
      sts = EsbGetNextItem(hCtx, ESB_LAPPDB_TYPE, AppDb)
      Debug.Print "EsbGetNextItem: sts = " & sts
      Debug.Print "App Name: "; AppDb.AppName
      Debug.Print "Db Name: "; AppDb.DbName
   Next

   'Reset password back to original
   NewPassword = "password"
   sts = EsbLoginSetPassword(hInst, Server, User, Password, NewPassword,
Items, hCtx)
   Debug.Print "EsbLoginSetPassword: sts = " & sts
End Sub

Sub ESB_SetActive()
   Dim AppName As String
   Dim DbName As String
   Dim pAccess As Integer
   Dim sts As Long

   'AppName = "Bugs"
   'DbName = "09129823"

   AppName = "vb"
   DbName = "Basic"

   sts = EsbSetActive(hCtx, AppName, DbName, pAccess)
   Debug.Print "EsbSetActive: sts = " & sts
End Sub

Sub ESb_GetStoresInfo() '(Chnl As String)
   Dim Object As ESB_OBJDEF_T

   Object.hCtx = hCtx
   Object.Type = ESB_OBJTYPE_OUTLINE
```

```
    Object.AppName = AppName
    Object.DbName = DbName
    Object.FileName = DbName
    Dim hMember As Long
    Dim ihMember As Long
    Dim MbrInfo As ESB_MBRINFO_T
    Dim Counts As ESB_MBRCOUNTS_T

    sts = EsbSetActive(hCtx, AppName, DbName, Access)

    Dim hMemberJan As Long
    Dim MbrChldCnt As Long
    Dim x As Integer
    Dim Parent As String
    Dim found As Boolean
    Dim img As Integer
    Dim Member As String
    Dim szAlias As String * ESB_MBRNAMELEN
    Dim Alias As String
    Dim levelnum As String
    Dim ShareStat As Integer

    Dim tLevelName As String * ESB_MBRNAMELEN
    Const AltGroup As String = "ALT_GROUP"
    'Dim LevelName As String * ESB_MBRNAMELEN
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)

    If sts = 0 Then
       sts = EsbOtlFindMember(hOutline, "JOHNSON, ROGER", hMemberJan)
       'sts = EsbOtlFindMember(hOutline, "GMM_A", hMemberJan)
       If hMemberJan = 0 Then
          sts = EsbOtlFindAlias(hOutline, "JOHNSON, ROGER", "default",
hMemberJan)
       End If
    End If

    If sts = 0 And hMemberJan <> 0 Then
       sts = EsbOtlGetMemberInfo(hOutline, hMemberJan, MbrInfo)
       MsgBox ("Member Name = " & MbrInfo.szMember)
       Member = MbrInfo.szMember
       levelnum = MbrInfo.usLevel
       ShareStat = MbrInfo.usShare
       MsgBox ("Shared Member = " & ShareStat)
    End If

    MbrChldCnt = MbrInfo.ulChildCount
    ' If ShareStat <> ESB_SHARE_SHARE Then

    'Do While x <= MbrChldCnt
    For x = 1 To MbrChldCnt
       If x = 1 Then
          sts = EsbOtlGetChild(hOutline, hMemberJan, hMember)
          'sts = EsbOtlGetMemberInfo(hOutline, hMember, MbrInfo)
          'MsgBox ("Child Member Name = " & MbrInfo.szMember)
       Else
          sts = EsbOtlGetNextSibling(hOutline, hMemberJan, hMember)
          ' sts = EsbOtlGetMemberInfo(hOutline, hMember, MbrInfo)
```

```
         ' MsgBox ("Sibling Member Name = " & MbrInfo.szMember)
      End If

   'Next

      sts = EsbOtlGetMemberInfo(hOutline, hMember, MbrInfo)
         MsgBox ("Sibling Member Name = " & MbrInfo.szMember)
         ' szAlias = ""
         'sts = EsbOtlGetMemberAlias(hOutline, hMember, "", szAlias)
         'sts = EsbOtlGetLevelName(hOutline, sRoot, MbrInfo.usLevel,
tLevelName)
         'If sts > 0 Then tLevelName = ""

      'Alias = sTrim(szAlias)
      'Member = sTrim(MbrInfo.szMember)

   Next
End Sub

Sub ESB_Logout()

   sts = EsbLogout(hCtx)
   'MsgBox ("EsbLogout = " & sts)
   Debug.Print "EsbLogout: sts = " & sts
End Sub

Sub ESB_Term()

   sts = EsbTerm(hInst)
   'MsgBox ("EsbTerm = " & sts)
   Debug.Print "EsbTerm: sts = " & sts
End Sub

Public Sub ESB_LROListObjects()
   Dim UserName As String * ESB_USERNAMELEN
   Dim listDate As Long
   Dim Items As Integer
   Dim Desc As ESB_LRODESC_API_T
   Dim i As Integer
   Dim j As Integer
   Dim CutOffDate As Date
   Dim MemberName As String * ESB_MBRNAMELEN

   Const ESB_REFERENCE_DATE = #1/1/1970#
   UserName = "essexer"
   CutOffDate = #9/21/2007#
   'CutOffDate = #1/2/1970#
   listDate = DateDiff("s", CutOffDate, ESB_REFERENCE_DATE)
   'listDate = DateDiff("s", ESB_REFERENCE_DATE, CutOffDate)
   'listDate = -1

   sts = EsbLROListObjects(hCtx, UserName, listDate, Items)
   Debug.Print "EsbLROListObjects: sts = " & sts

   Debug.Print "Number of LRO(s): " & Items

   If sts = 0 Then
```

```
      For i = 1 To Items

         Debug.Print "LRO # " & i; ":"

         sts = EsbGetNextItem(hCtx, ESB_LRO_TYPE, Desc)
         Debug.Print "EsbGetNextItem: sts = " & sts
         Debug.Print "Object Type: " & Desc.ObjType
         Select Case (Desc.ObjType)
            Case 0
                Debug.Print "Cell notes: " & Desc.note
            Case 1
                Debug.Print "Object Name: " & Desc.lroInfo.ObjName
                Debug.Print "Object Description: " & Desc.lroInfo.objDesc
            Case 2
                Debug.Print "Object Name: " & Desc.lroInfo.ObjName
                Debug.Print "Object Description: " & Desc.lroInfo.objDesc
         End Select
         Debug.Print "Member Combination:"
         For j = 1 To Desc.memCount
            sts = EsbLROGetMemberCombo(hCtx, j, MemberName)
            Debug.Print "    " & MemberName
         Next j

      Next i
   End If
End Sub

Sub ESB_SetUser()
   Dim sts As Long
   Dim UserInfo As ESB_USERINFO_T

   UserInfo.Name = "Test"
   UserInfo.Type = ESB_TYPE_USER
   UserInfo.Access = ESB_ACCESS_SUPER
   UserInfo.MaxAccess = ESB_ACCESS_SUPER
   UserInfo.PwdChgNow = ESB_TRUE

   sts = EsbSetUser(hCtx, UserInfo)
   Debug.Print "EsbSetUser: sts = " & sts
End Sub

Sub ESB_GetUser()
   Dim sts As Long
   Dim User As String
   Dim UserInfo As ESB_USERINFO_T

   User = "Test"
   '************************
   ' Get User Info structure
   '************************
   sts = EsbGetUser(hCtx, User, UserInfo)
   Debug.Print "EsbGetUser: sts = " & sts
End Sub

Public Sub ESB_LROPurgeObjects()
   Dim UserName As String * ESB_USERNAMELEN
   Dim purgeDate As Long
```

```
   Dim Items As Integer
   Dim Desc As ESB_LRODESC_API_T
   Dim CutOffDate As Date
   Dim i As Integer
   Const ESB_REFERENCE_DATE = #1/1/1970#

   UserName = "essexer"
   CutOffDate = #9/21/2007#
   purgeDate = DateDiff("s", ESB_REFERENCE_DATE, CutOffDate) 'bug
8-651484045
   'purgeDate = DateDiff("s", CutOffDate, ESB_REFERENCE_DATE)
   'purgeDate = -1

   sts = EsbLROPurgeObjects(hCtx, UserName, purgeDate, Items)
   Debug.Print "EsbLROPurgeObjects: sts = " & sts

   If sts = 0 Then
      For i = 1 To Items
         '*****************************
         '* Get the next LRO description
         '* item from the list
         '*****************************
         sts = EsbGetNextItem(hCtx, ESB_LRO_TYPE, Desc)
         Debug.Print "EsbGetNextItem: sts = " & sts
      Next i
   End If
End Sub

Sub ESB_CreateGroup()
   Dim sts As Long
   Dim GroupName As String

   GroupName = "PowerUsers"
  sts = EsbCreateGroup(hCtx, GroupName)
   Debug.Print "EsbCreateGroup: sts = " & sts
End Sub

Sub ESB_GetDatabaseInfo()
   Dim sts As Long
   Dim AppName As String
   Dim DbName As String
   Dim Items As Integer
   Dim N As Integer
   Dim DbInfo As ESB_DBINFO_T
   Dim DbReqInfo As ESB_DBREQINFO_T

   AppName = "Sample"
   DbName = "Basic"
  sts = EsbGetDatabaseInfo(hCtx, AppName, DbName, DbInfo, Items)
   Debug.Print "EsbGetDatabaseInfo: sts = " & sts
   Debug.Print "DbInfo.status: " & DbInfo.Status

   If sts = 0 Then
      For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_DBREQINFO_TYPE, DbReqInfo)
         Debug.Print "EsbGetNextItem: sts = " & sts
      Next
```

```
   End If
End Sub

Sub ESB_GetDatabaseAccess()
   Dim Items As Integer
   Dim AppName As String
   Dim DbName As String
   Dim User As String
   Dim UserDb As ESB_USERDB_T
   Dim sts As Long

   AppName = "Sample"
   DbName = "Basic"

   User = "user1"
  sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
   Debug.Print "EsbGetDatabaseAccess: sts = " & sts
   For N = 1 To Items
    sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
     Debug.Print "EsbGetNextItem: sts = " & sts
     Debug.Print "User: " & User
     Debug.Print "Access: " & UserDb.Access
   Next

   User = "user2"
  sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
   Debug.Print "EsbGetDatabaseAccess: sts = " & sts
   For N = 1 To Items
    sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
     Debug.Print "EsbGetNextItem: sts = " & sts
     Debug.Print "User: " & User
     Debug.Print "Access: " & UserDb.Access
   Next

   User = "user3"
  sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
   Debug.Print "EsbGetDatabaseAccess: sts = " & sts
   For N = 1 To Items
    sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
     Debug.Print "EsbGetNextItem: sts = " & sts
     Debug.Print "User: " & User
     Debug.Print "Access: " & UserDb.Access
   Next

   User = "user4"
  sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
   Debug.Print "EsbGetDatabaseAccess: sts = " & sts
   For N = 1 To Items
    sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
     Debug.Print "EsbGetNextItem: sts = " & sts
     Debug.Print "User: " & User
     Debug.Print "Access: " & UserDb.Access
   Next

   User = "user5"
  sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
   Debug.Print "EsbGetDatabaseAccess: sts = " & sts
```

```
   For N = 1 To Items
    sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
     Debug.Print "EsbGetNextItem: sts = " & sts
     Debug.Print "User: " & User
     Debug.Print "Access: " & UserDb.Access
   Next

   User = "user6"
  sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
   Debug.Print "EsbGetDatabaseAccess: sts = " & sts
   For N = 1 To Items
    sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
     Debug.Print "EsbGetNextItem: sts = " & sts
     Debug.Print "User: " & User
     Debug.Print "Access: " & UserDb.Access
   Next
End Sub

Sub ESB_GetDatabaseStats()
   Dim Items As Integer
   Dim AppName As String
   Dim DbName As String
   Dim DbStats As ESB_DBSTATS_T
   Dim DimStats As ESB_DIMSTATS_T
   Dim sts As Long
   AppName = "Sample"
   DbName = "Basic"
   sts = EsbGetDatabaseStats(hCtx, AppName, DbName, DbStats, Items)
   Debug.Print "EsbGetDatabaseStats: sts = " & sts
   'MsgBox ("cluster = " & DbStats.ClusterRatio)
   For N = 1 To Items
      sts = EsbGetNextItem(hCtx, ESB_DBSTATS_TYPE, DbStats)
   Next
End Sub

Public Sub ESB_LROAddObject()
   Dim Desc As ESB_LRODESC_API_T
   Dim memCount As Long
   Dim memComb As String
   Dim opt As Integer
   Dim i As Integer

   memCount = 5
   memComb = "Year" & vbCrLf & "Product" & vbCrLf & _
      "Market" & vbCrLf & "Measures" & vbCrLf & "Scenario"
   Desc.UserName = "essexer"

   Desc.ObjType = ESB_LROTYPE_CELLNOTE_API
   Desc.note = "Cell note"
   opt = ESB_NOSTORE_OBJECT_API
   sts = EsbLROAddObject(hCtx, memCount, memComb, opt, Desc)
   Debug.Print "EsbLROAddObject: sts = " & sts

   Desc.ObjType = ESB_LROTYPE_WINAPP_API
   Desc.lroInfo.ObjName = "c:\hyperion\essbase95\bin\essbase.exe"
   Desc.lroInfo.objDesc = "Essbase executable."
   opt = ESB_STORE_OBJECT_API
```

```
   sts = EsbLROAddObject(hCtx, memCount, memComb, opt, Desc)
   Debug.Print "EsbLROAddObject: sts = " & sts

   Desc.ObjType = ESB_LROTYPE_URL_API
   Desc.lroInfo.ObjName = "www.oracle.com"
   Desc.lroInfo.objDesc = "Oracle homepage"
   opt = ESB_NOSTORE_OBJECT_API
   sts = EsbLROAddObject(hCtx, memCount, memComb, opt, Desc)
   Debug.Print "EsbLROAddObject: sts = " & sts

    Desc.ObjType = ESB_LROTYPE_CELLNOTE_API
    Desc.note = "Cell note 2"
    opt = ESB_NOSTORE_OBJECT_API
   sts = EsbLROAddObject(hCtx, memCount, memComb, opt, Desc)
    Debug.Print "EsbLROAddObject: sts = " & sts
End Sub

Public Sub ESB_LROGetCatalog()

   Dim Desc As ESB_LRODESC_API_T
   Dim Items As Integer
   Dim memCount As Long
   Dim memComb As String
   Dim i As Integer

   memCount = 5
   memComb = "Qtr1" & vbCrLf & "Profit" & vbCrLf & _
             "100" & vbCrLf & "East" & vbCrLf & "Scenario"
   'memComb = "Jan" & vbCrLf & "Sales" & _
   '          "Cola" & vbCrLf & "Utah" & _
   '          "Actual"

  sts = EsbLROGetCatalog(hCtx, memCount, memComb, Items)
   Debug.Print "EsbLROGetCatalog: sts = " & sts

   If sts = 0 Then
      For i = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_LRO_TYPE, Desc)
         Debug.Print "Desc.ObjType = " & Desc.ObjType
         Debug.Print "Desc.note = " & Desc.note
         Debug.Print "Desc.lroInfo.objDesc = " & Desc.lroInfo.objDesc
         Debug.Print "Desc.lroInfo.objName = " & Desc.lroInfo.ObjName
      Next i
   End If
End Sub

Sub ESB_CopyObject()
   Dim sts As Long
   Dim SrcApp As String
   Dim SrcDb As String
   Dim SrcObj As String
   Dim DestApp As String
   Dim DestDb As String
   Dim DestObj As String

   SrcApp = "Sample"
   SrcDb = "Basic"
```

```
   SrcObj = "Basic"

   DestApp = "Sample"
   DestDb = "Basic"
   DestObj = "Basic1"
   ObjType = ESB_OBJTYPE_OUTLINE

  sts = EsbCopyObject(hCtx, hDestCtx, ObjType, SrcApp, DestApp, _
   SrcDb, DestDb, SrcObj, DestObj)
   Debug.Print "EsbCopyObject: sts = " & sts
End Sub

Sub ESB_GetAssociatedAttributesInfo()
   Dim sts As Long
   Dim MbrName As String
   Dim AttrDimName As String
   Dim Count As Long
   Dim Attribinfo As ESB_ATTRIBUTEINFO_T
   Dim index As Integer
   Dim tempstring As String

   'MbrName = InputBox("Base member name", "Base Member Name")
   'AttrDimName = InputBox("Attribute Dimension Name (Optional)",
"Attribute Dimension Name")

   MbrName = "em41666"
   AttrDimName = "Job Start Date"
  sts = EsbGetAssociatedAttributesInfo(hCtx, MbrName, AttrDimName, Count)
   Debug.Print "EsbGetAssociatedAttributesInfo: sts = " & sts

   Debug.Print "Associated Attr info for: " & MbrName

   For index = 1 To Count
     sts = EsbGetNextItem(hCtx, ESB_ATTRIBUTEINFO_TYPE, Attribinfo)
      'Debug.Print "Dim Name: " & Attribinfo.DimName
      Debug.Print "Attribute Dim Name: " & Attribinfo.DimName
      Debug.Print "Attribute Mbr Name: " & Attribinfo.MbrName

      ' NOTE: use of select case statement to discern (and act upon) type
of attribute returned
      Select Case VarType(Attribinfo.Attribute)
         Case vbDouble
            Debug.Print "Data Type    : Numeric(Double)"
            Debug.Print "Data Value   : " & Attribinfo.Attribute
            Debug.Print ""
         Case vbBoolean
            Debug.Print "Data Type    : Boolean"
            Debug.Print "Data Value   : " & Attribinfo.Attribute
            Debug.Print ""
         Case vbDate
            Debug.Print "Data Type    : Date"
            Debug.Print "Data Value   : " & Attribinfo.Attribute
            Debug.Print ""
         Case vbString
            Debug.Print "Data Type    : String"
            Debug.Print "Data Value   : " & Attribinfo.Attribute
            Debug.Print ""
```

```
         End Select
      Debug.Print ""
   Next index
End Sub

Sub ESB_ListConnections()
   Dim Items As Integer
   Dim UserInfo As ESB_USERINFO_T
   Dim sts As Long

   sts = EsbListConnections(hCtx, Items)
   Debug.Print "EsbListConnections: sts = " & sts

   For N = 1 To Items
     sts = EsbGetNextItem(hCtx, ESB_USERINFO_TYPE, UserInfo)
      Debug.Print "EsbGetNextItem: sts = " & sts
   Next
End Sub

Sub ESB_ListRequests()
    Dim Items As Integer
    Dim ReqInfo As ESB_REQUESTINFO_T
    Dim sts As Long

    sts = EsbListRequests(hCtx, UserName, AppName, DbName, Items)
    Debug.Print "EsbListRequests: sts = " & sts

    For N = 1 To Items
     sts = EsbGetNextItem(hCtx, ESB_REQUESTINFO_TYPE, ReqInfo)
      Debug.Print "EsbGetNextItem: sts = " & sts
      Debug.Print "AppName: " & ReqInfo.AppName
      Debug.Print "DbName: " & ReqInfo.DbName
      Debug.Print "DbRequestCode: " & ReqInfo.DbRequestCode
      Debug.Print "LoginID: " & ReqInfo.LoginId
      Debug.Print "LoginSourceMachine: " & ReqInfo.LoginSourceMachine
      Debug.Print "RequestString: " & ReqInfo.RequestString
      Debug.Print "State: " & ReqInfo.State
      Debug.Print "TimeStarted: " & ReqInfo.TimeStarted
      Debug.Print "Username: " & ReqInfo.UserName
    Next
End Sub

Sub ESB_AddToGroup()
    Dim sts As Long
    Dim GroupName As String
    Dim User As String

    GroupName = "Group1"
    User = "user1"
    sts = EsbAddToGroup(hCtx, GroupName, User)
    Debug.Print "EsbAddToGroup sts: " & sts
End Sub

Sub ESB_GetGroupList()
   Dim Items As Integer
   Dim Group As String
   Dim GroupName As String * ESB_USERNAMELEN
```

```
   Dim sts As Long

   Group = "group1"
   sts = EsbGetGroupList(hCtx, Group, Items)
   Debug.Print "EsbGetGroupList: sts = " & sts

   For N = 1 To Items
    sts = EsbGetNextItem(hCtx, ESB_GROUPNAME_TYPE, ByVal GroupName)
     Debug.Print "EsbGetGroupList: sts = " & sts
     Debug.Print "User Name = " & GroupName
     MsgBox ("User Name = " & GroupName)
   Next
End Sub

Sub ESB_GetDatabaseState()
   Dim sts As Long
   Dim AppName As String
   Dim DbName As String
   Dim DbState As ESB_DBSTATE_T
   AppName = "Sample"
   DbName = "Basic"

  sts = EsbGetDatabaseState(hCtx, AppName, DbName, DbState)
   Debug.Print "EsbGetDatabaseState: sts = " & sts
End Sub

Sub ESB_PartitionReadDefFile()
   Dim sts As Long
   Dim iFileHandle As Long
   Dim pszFileName As String
   Dim DdbCtx As Long
   Dim pDdbCtx As Long
   Dim partDefined As ESB_PART_DEFINED_T
   Dim partInfo As ESB_PART_INFO_T
   Dim partConnectInfo As ESB_PART_CONNECT_INFO_T

   pszFileName = "c:\\hyperion\\essbase95\\app\\sample\\basic\\basic.ddb"
   ' Not public
   sts = EsbPartitionOpenDefFile(hCtx, pszFileName, iFileHandle, pDdbCtx)
   Debug.Print "EsbPartitionOpenDefFile sts: " & sts
   sts = EsbGetNextItem(hCtx, ESB_PART_DEFINED_T, partDefined)
   Debug.Print "EsbGetNextItem sts: " & sts

   sts = EsbGetNextItem(hCtx, ESB_PART_INFO_T, partInfo)
   Debug.Print "EsbGetNextItem sts: " & sts

   sts = EsbGetNextItem(hCtx, ESB_PART_CONNECT_INFO_T, partConnectInfo)
   Debug.Print "EsbGetNextItem sts: " & sts

   sts = EsbGetNextItem(hCtx, ESB_PART_T, partInfo)
   Debug.Print "EsbGetNextItem sts: " & sts

   sts = EsbGetNextItem(hCtx, ESB_PART_T, partInfo)
   Debug.Print "EsbGetNextItem sts: " & sts

   sts = EsbPartitionReadDefFile(hCtx, iFileHandle, DdbCtx)
   Debug.Print "EssPartitionReadDefFile sts: " & sts
```

```
   sts = EsbGetNextItem(hCtx, ESB_PART_DEFINED_T, partInfo)
   Debug.Print "EsbGetNextItem sts: " & sts

   sts = EsbPartitionCloseDefFile(hCtx, iFileHandle)
   Debug.Print "EssPartitionCloseDefFile sts: " & sts

   sts = EsbPartitionFreeDefCtx(hCtx, pDdbCtx)
   Debug.Print "EssPartitionFreeDefCtx sts: " & sts
End Sub

Public Sub ESB_PartitionWriteDefFile()
   Dim SelectPartition As ESB_PARTSLCT_T
   Dim Partition As ESB_PART_INFO_T
   Dim Items As Integer
   Dim i As Integer

   Dim FileName As String
   Dim HostDatabase As ESB_PART_CONNECT_INFO_T
   Dim FileHandle1 As Long
   Dim FileHandle2 As Long
   Dim DdbCtxHandle1 As Long
   Dim DdbCtxHandle2 As Long
   HostDatabase.AppName = "Sample"
   HostDatabase.DbName = "Basic"
   HostDatabase.HostName = "Localhost"

   FileName = "C:\Hyperion\essbase95\app\Sample\Basic\Basic.ddb"
   sts = EsbPartitionOpenDefFile(hCtx, FileName, FileHandle1,
DdbCtxHandle1)
   Debug.Print "EsbPartitionOpenDefFile sts: " & sts

   sts = EsbPartitionReadDefFile(hCtx, FileHandle1, DdbCtxHandle1)
   Debug.Print "EsbPartitionReadDefFile sts: " & sts

   FileName = "C:\Hyperion\essbase95\app\Sample\Basic\Basic.ddn"
   sts = EsbPartitionNewDefFile(hCtx, FileName, HostDatabase, FileHandle2,
DdbCtxHandle2)
   Debug.Print "EsbPartitionNewDefFile sts: " & sts

   sts = EsbPartitionWriteDefFile(hCtx, FileHandle2, DdbCtxHandle1)
   Debug.Print "EsbPartitionWriteDefFile sts: " & sts
   'Debug.Print "Description: " & DdbCtxHandle1.

   sts = EsbPartitionCloseDefFile(hCtx, FileHandle1)
   Debug.Print "EsbPartitionCloseDefFile sts: " & sts

   sts = EsbPartitionCloseDefFile(hCtx, FileHandle2)
   Debug.Print "EsbPartitionCloseDefFile sts: " & sts

   sts = EsbPartitionReplaceDefFile(hCtx) 'It is assumed that .ddn file
must be present in database
   Debug.Print "EsbPartitionReplaceDefFile sts: " & sts

   sts = EsbPartitionFreeDefCtx(hCtx, DdbCtxHandle2)
   Debug.Print "EsbPartitionFreeDefCtx sts: " & sts

   sts = EsbPartitionFreeDefCtx(hCtx, DdbCtxHandle2)
```

```
   Debug.Print "EsbPartitionFreeDefCtx sts: " & sts

End Sub

Public Sub ESB_PartitionReplaceDefFile()

   Dim SelectPartition As ESB_PARTSLCT_T
   Dim Items As Integer
   Dim i As Integer

   Dim FileName As String
   Dim HostDatabase As ESB_PART_CONNECT_INFO_T
   Dim FileHandle1 As Long
   Dim FileHandle2 As Long

   'FileName = "C:\Hyperion\essbase95\app\Sample\Basic\Basic.ddn"
   'sts = EsbPartitionOpenDefFile(hCtx, FileName, FileHandle1,
DdbCtxHandle1)
   'Debug.Print "EsbPartitionOpenDefFile sts: " & sts

   FileName = "C:\Hyperion\essbase95\app\Sample\Basic\Basic.ddb"
   HostDatabase.AppName = "samppart"
   HostDatabase.DbName = "Company"
   HostDatabase.HostName = "LocalHost"

   sts = EsbPartitionNewDefFile(hCtx, FileName, HostDatabase, FileHandle2,
DdbCtxHandle2)
   Debug.Print "EsbPartitionNewDefFile sts: " & sts

   sts = EsbPartitionWriteDefFile(hCtx, FileHandle2, DdbCtxHandle2)
   Debug.Print "EsbPartitionWriteDefFile sts: " & sts

   sts = EsbPartitionReplaceDefFile(hCtx) 'It is assumed that .ddn file
must be present in database
   Debug.Print "EsbPartitionReplaceDefFile sts: " & sts
End Sub

Public Sub ESB_PartitionValidateDefinition()

   pRemoteDDBFileName = "east"
   pSelectVerify.usLoc = ESB_FILE_SERVER
   pszDefFile = "east"

   pSelectVerify.Partition.usType = ESB_PARTITION_OP_TRANSPARENT
   pSelectVerify.Partition.Direction = ESB_PARTITION_DATA_SOURCE
   pSelectVerify.Partition.HostDatabase.HostName = "Localhost"
   pSelectVerify.Partition.HostDatabase.AppName = "Samppart"
   pSelectVerify.Partition.HostDatabase.DbName = "Company"

   'sts = EsbPartitionValidateDefinition(hCtx, pSelectVerify, pszDefFile,
pInvalidComponentCount, InValidComponentsHandle, pRemoteDDBFileName)
   Debug.Print "EsbPartitionValidateDefinition sts: " & sts
End Sub

Sub ESB_PartitionValidateLocal()

   sts = EsbPartitionValidateLocal(hCtx, ValidationFlag)
```

```
   Debug.Print "EsbPartitionValidateDefinition sts: " & sts
End Sub

Public Sub ESB_PartitionReadOtlChangeFile()
   Dim pszChgFileName As String
   'Dim MetaChangeRecord1 As ESB_PARTOTL_READ_T
   'Dim MetaChangeRecord2 As ESB_READ_T

   PartQuery.TimeStamp = DateDiff("s", #1/1/1970#, #6/18/1997#)
   PartQuery.DimFilter = ESB_PARTITION_OTLDIM_ALL
   PartQuery.MbrFilter = ESB_PARTITION_OTLMBR_ALL
   PartQuery.MbrAttrFilter = ESB_PARTITION_OTLMBRATTR_ALL
   pszChgFileName = "c:\hyperion\Essbase95\app\Samppart\Company
\ess00008.chg"

   'sts = EsbPartitionReadOtlChangeFile(hCtx, pszChgFileName, PartQuery,
MetaChangeReadHandle, SourceTime)
   Debug.Print "EsbPartitionReadOtlChangeFile sts: " & sts

   'sts = EsbGetNextItem(hCtx, ESB_PARTOTL_READ_T, MetaChangeRecord1)
   'Debug.Print "EsbGetNextItem sts: " & sts

   'sts = EsbGetNextItem(hCtx, ESB_PARTOTL_READ_T, MetaChangeRecord2)
   'Debug.Print "EsbGetNextItem sts: " & sts

   sts = EsbPartitionFreeOtlChanges(hCtx, MetaChangeReadHandle)
   Debug.Print "EsbPartitionFreeOtlChanges sts: " & sts
End Sub

Sub ESB_CreateLocalContext()
   Dim sts As Long
   Dim User As String
   Dim Password As String
   Dim hCtx As Long

   '*********************
   ' Create Local Context
   '*********************
  sts = EsbCreateLocalContext(hInst, User, Password, hCtx)
End Sub

Sub ESB_Import()
   Dim sts          As Long
   Dim Rules        As ESB_OBJDEF_T
   Dim Data         As ESB_OBJDEF_T
   Dim User         As ESB_MBRUSER_T
   Dim ErrorName    As String
   Dim AbortOnError As Integer
   Dim hLocalCtx As Long

   '************************************************************
   ' Need to create a local context, if files are not on the server
   '************************************************************
  sts = EsbCreateLocalContext(hInst, "", "", hLocalCtx)
   Debug.Print "EsbCreateLocalContext sts: " & sts
   Data.hCtx = hLocalCtx
   Data.Type = ESB_OBJTYPE_TEXT
```

```
    Data.AppName = ""
    Data.DbName = ""
    Data.FileName = "F:\\testArea\\VBAPI\\calcdat.txt"

    '*********************************
    ' Rules file resides at the server
    '*********************************
    'Rules.hCtx = hCtx
    'Rules.Type = ESB_OBJTYPE_RULES
    'Rules.AppName = "Demo"
    'Rules.DbName = "Basic"
    'Rules.FileName = "Test"

    '*********************************
    ' Data file resides at the server
    '*********************************
    'Data.hCtx = hCtx
    'Data.Type = ESB_OBJTYPE_TEXT
    'Data.AppName = "Demo"
    'Data.DbName = "Basic"
    'Data.FileName = "Data"

    '*********************************
    ' Specify file to redirect errors
    ' to if any
    '*********************************
    ErrorName = "IMPORT.ERR"

    '************************
    ' Abort on the first error
    '************************
    AbortOnError = ESB_YES

    '*******
    ' Import
    '*******
   sts = EsbImport(hCtx, Rules, Data, User, ErrorName, AbortOnError)
    Debug.Print "EsbImport sts: " & sts
End Sub

Sub ESB_VerifyFilter()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim Row As String

    AppName = "Sample"
    DbName = "Basic"

   sts = EsbVerifyFilter(hCtx, AppName, DbName)
    Debug.Print "EsbVerifyFilter sts: " & sts

    ' Initialize Filter Row
    Row = "@IDESCENDANTS(Scenario)"
   sts = EsbVerifyFilterRow(hCtx, Row) ' Initialize Filter Row
    Debug.Print "EsbVerifyFilterRow sts: " & sts
```

```
   Row = "@IDESCENDANTS(AAAA)"
  sts = EsbVerifyFilterRow(hCtx, Row)
   Debug.Print "EsbVerifyFilterRow sts: " & sts

  sts = EsbVerifyFilterRow(hCtx, ByVal 0&)
   Debug.Print "EsbVerifyFilterRow sts: " & sts
End Sub

Sub Test()
    strComputer = "."
    Const ForReading = 1
    Const ForWriting = 2
    Const ForAppending = 8
    '================================================================
    Const Data_Path = "F:\Testarea\temp\"
    Const FileName = "process.txt"

    Set fso = CreateObject("Scripting.FileSystemObject")
    If Not fso.FileExists(Data_Path & FileName) Then
     Set f = fso.OpenTextFile(Data_Path & FileName, 2, True)
    Else
     Set f = fso.OpenTextFile(Data_Path & FileName, 8)
    End If

    Set objWMIService = GetObject("winmgmts:" &
"{impersonationLevel=impersonate}!\\" & strComputer & "\root\cimv2")
    Set colProcessList = objWMIService.ExecQuery("Select * from
Win32_Process")
    For Each objProcess In colProcessList
        f.WriteLine "Process " & objProcess.Name
    Next
End Sub

Sub ESB_CreateGLDrillThru()
   Dim sts          As Long
   Dim url          As ESB_DURLINFO_T
   Dim cppDrillRegions(0 To 1) As String

   '************************************************************
   ' Need to create a local context, if files are not on the server
   '************************************************************
   url.bIsLevel0 = 0

   cppDrillRegions(0) = "sales"
   cppDrillRegions(1) = "cogs"
   url.cpURLXML = "Testing"
   url.cpURLName = "VB URL7"
   url.iURLXMLSize = 8

   sts = EsbCreateDrillThruURL(hCtx, cppDrillRegions, url)

   Debug.Print "EsbCreateDrillThruURL sts: " & sts

End Sub

Sub ESB_UpdateGLDrillThru()
   Dim sts                      As Long
```

```
   Dim url                        As ESB_DURLINFO_T
   Dim cppDrillRegions(0 To 1)    As String
   Dim bMerge                     As Integer

   '*************************************************************
   ' Need to create a local context, if files are not on the server
   '*************************************************************
   url.bIsLevel0 = 0
   bMerge = ESB_TRUE

   cppDrillRegions(0) = "qtr1"
   url.cpURLXML = "Testing"
   url.cpURLName = "VB URL7"
   url.iURLXMLSize = 8

   sts = EsbUpdateDrillThruURL(hCtx, cppDrillRegions, url, bMerge)

   Debug.Print "EsbUpdateDrillThruURL sts: " & sts
End Sub

Sub ESB_DeleteGLDrillThru()
   Dim URLName      As String

   URLName = "VB URL7"
   sts = EsbDeleteDrillThruURL(hCtx, URLName)

   Debug.Print "EsbDeleteDrillThruURL sts: " & sts
End Sub

Sub ESB_GetGLDrillThru()
   Dim URLName      As String
   Dim url          As ESB_DURLINFO_T
   Dim intX         As Integer
   Dim cppDrillRegions  As Variant

   URLName = "VB URL2"
   sts = EsbGetDrillThruURL(hCtx, URLName, url, cppDrillRegions)

   Debug.Print "EsbGetDrillThruURL sts: " & sts

   If sts = 0 Then
    Debug.Print "URL Name: " & url.cpURLName
    Debug.Print "URL XML: " & url.cpURLXML

    For intX = LBound(cppDrillRegions) To UBound(cppDrillRegions)

    Debug.Print "URL Region: " & cppDrillRegions(intX)

    Next
   End If
End Sub

Sub ESB_ListGLDrillThru()
   Dim intX         As Integer
   Dim URLNames  As Variant

   sts = EsbListDrillThruURLs(hCtx, URLNames)
```

```
   If sts = 0 Then
       Debug.Print "EsbListDrillThruURL sts: " & sts

       For intX = LBound(URLNames) To UBound(URLNames)

       Debug.Print "URL Name: " & URLNames(intX)

       Next
   End If
End Sub

Sub ESB_GetCellDrillThruReports()
   Dim intX         As Integer
   Dim mbrs(0 To 4) As String
   Dim pURLXMLLens  As Variant
   Dim pURLXMLs  As Variant

   mbrs(0) = "sales"
   mbrs(1) = "jan"
   mbrs(2) = "New York"
   mbrs(3) = "actual"
   mbrs(4) = "100-10"

   sts = EsbGetCellDrillThruReports(hCtx, mbrs, pURLXMLLens, pURLXMLs)

   If sts = 0 Then

       Debug.Print "EsbGetCellDrillThruReports sts: " & sts

       For intX = LBound(pURLXMLLens) To UBound(pURLXMLLens)

       Debug.Print "URL XML: " & intX
       Debug.Print "URL XML Len: " & pURLXMLLens(intX)
       Debug.Print "URL XML String: " & pURLXMLs(intX)

       Next
   End If

   mbrs(0) = "profit"
   sts = EsbGetCellDrillThruReports(hCtx, mbrs, pURLXMLLens, pURLXMLs)

   If sts = 0 Then

       Debug.Print "EsbGetCellDrillThruReports sts: " & sts

       For intX = LBound(pURLXMLLens) To UBound(pURLXMLLens)

       Debug.Print "URL XML: " & intX
       Debug.Print "URL XML Len: " & pURLXMLLens(intX)
       Debug.Print "URL XML String: " & pURLXMLs(intX)

       Next
   End If
End Sub

Sub Main()
```

```
    'Test
    ESB_Init
    'ESB_CreateLocalContext
    'ESB_AutoLogin
    ESB_Login
    'ESB_LoginSetPassword
    ESB_SetActive
    ESB_CreateGLDrillThru
    ESB_UpdateGLDrillThru
    ESB_GetGLDrillThru
    ESB_ListGLDrillThru
    ESB_GetCellDrillThruReports
    ESB_DeleteGLDrillThru
    'ESB_GetGLDrillThru
    'ESB_ListGLDrillThru
    ESB_GetCellDrillThruReports
    'ESB_SetUser
    'ESB_GetUser
    'ESB_GetMessage
    'ESB_Import
    'ESB_GetVersion
    'ESB_GetDatabaseInfo
    'ESB_GetDatabaseState
    'ESB_GetDatabaseStats
    'ESB_GetDatabaseAccess
    'ESB_GetGroupList
    'ESB_ListConnections
    'ESB_ListRequests
    'ESB_GetAssociatedAttributesInfo
    'ESb_GetStoresInfo
    'ESB_OtlGetMemberAlias
    'ESB_AddAliasCombination
    'ESB_CreateGroup
    'ESB_LROAddObject
    'ESB_LROGetCatalog
    'ESB_LROListObjects
    'ESB_LROPurgeObjects

   'ESB_CopyObject
   'ESB_PartitionReadDefFile
   'ESB_PartitionWriteDefFile
   'ESB_PartitionReplaceDefFile
   'ESB_PartitionValidateDefinition
   'ESB_PartitionValidateLocal
   'ESB_PartitionReadOtlChangeFile

   'ESB_AddToGroup
   'ESB_GetGroupList
   'ESB_VerifyFilter
   ESB_Logout
   ESB_Term
End Sub
```

Administration Services

The following Administration Services help topics are for the drill-through to Oracle applications feature.

- "Managing Drill-through Definitions" on page 50
- "Edit Drill-Through Definitions Dialog Box" on page 51

Managing Drill-through Definitions

➤ To manage drill-through definitions:

1 **In Enterprise View, right-click on a database.**

2 **Select Edit, and then Drill-through definitions.**

3 **Perform an action:**

- Add a definition
- Modify a definition
- Delete a definition

➤ To add or modify a definition:

1 **Perform an action:**

- To add a definition, in **Definitions**, select **Click here to add a definition**.
- To modify a definition, select it in **Definitions**.

2 **Optional: If creating a definition, enter a URL name.**

3 **Perform an action:**

- In **XML Contents**, enter an XML script.
- Click **Load XML from file**, and select a file containing an XML script.
- Click **Save XML to file** to save the XML script.
- Click **Export XML** to export the XML script.

4 **Add one or more regions to include in the definition:**

a. In the outline tree, double-click member names to insert them into the formula at the text marker position. You can perform Find Members operations to locate members containing specific text.

b. **Optional:** To view alias names in the outline tree, select **Use aliases** and select an alias table from the list.

c. In the Commands and functions tree, double-click an operator or function. The selected operator or function is inserted in the text area at the text-marker position. Select **Insert arguments** to include arguments in the text area as the command or function is inserted.

d. **Optional:** To include only level-0 members, select **Level zero members only**.

5 **Click Save, and then Close.**

➤ To delete a definition:

1. In **Definitions**, select a drill-through definition.
2. Click **Delete definition**.

Edit Drill-Through Definitions Dialog Box

You use the Edit Drill-Through Definitions Dialog Box to add, modify, or delete drill-through definitions.

Select a drill-through definition from **Definitions** to edit or delete it, or select **Click here to add a new definition**.

Information that can be edited in the Edit Drill-through Definitions dialog box for a drill-through definition:

- URL name—Name of the drill-through definition as it appears in client applications
- XML Contents—Script defining how the client application retrieves data from Essbase
- Regions—Database slices containing information available to the client application.

You add members to **Regions** by double-clicking them in the member tree. You apply commands and functions to members in **Regions** by double-clicking them in **Commands and Functions**. Optionally, you use aliases and have Administration Services insert function arguments automatically by selecting **Use aliases** or **Insert arguments**.

You load or export XML scripts for a definition by selecting **Load XML from file** or **Export XML**.

FDM Enhancements

After data or metadata is loaded using Oracle Hyperion Financial Data Quality Management, Fusion Edition, planners can drill through to the FDM source details of cell data from within the Oracle Hyperion Planning, Fusion Edition data form. With this release, FDM is automatically enabled; administrators do not need to enable it as an application setting. Users can also drill-through to FDM from Smart View or Financial Reporting.

ERP Integrator

Oracle Hyperion Financial Data Quality Management ERP Integration Adapter for Oracle Applications is a module of FDM that enables you to:

- Integrate metadata and data from an Enterprise Resource Planning (ERP) source system into an Oracle Hyperion EPM target application.
- Drill through from the EPM application (Oracle Hyperion Financial Management, Fusion Edition or Planning through web forms, Smart View or Oracle Hyperion Financial Reporting, Fusion Edition) and view details in the ERP source system.

For more information about ERP Integrator, see the *Oracle Hyperion Financial Data Quality Management ERP Integration Adapter for Oracle Applications Administrator's Guide*.

For information about the versions of Oracle E-Business Suite and PeopleSoft for which general ledger data is supported, see the *Oracle Hyperion Enterprise Performance Management System Certification Matrix* at http://www.oracle.com/technology/products/bi/hyperion-supported-platforms.html.

Smart View Enhancements

Smart View's drill-through capabilities have been enhanced to include ERP Integrator, Oracle General Ledger, and FDM:

- If you are connected to Planning or Financial Management via Smart View, you can use the drill-through capabilities of Oracle Hyperion Smart View for Office, Fusion Edition to drill through your Planning or Financial Management application to detailed data in Oracle Hyperion Financial Data Quality Management ERP Integration Adapter for Oracle Applications or Oracle Hyperion Financial Data Quality Management, Fusion Edition data sources.
- For applications created in Administration Services, you can drill through to Oracle General Ledger.

For applications created in Essbase Studio or Oracle Essbase Integration Services, you can continue to drill through to relational databases. For applications created in Oracle Essbase Studio, you can also drill through to administrator-configured URLs.

Note: To enable drill-through, all data source providers are front-ended with a proxy server. See the *Oracle Hyperion Enterprise Performance Management System Installation and Configuration Guide*.

Calculation Manager Enhancements

For this release, Hyperion Calculation Manager supports users of Essbase block storage applications. You can use Calculation Manager to design, launch, and administer Essbase business rules. These are the new features for this release of Calculation Manager:

Support for Calculation Manager in Classic Applications and Essbase Block Storage Applications

Calculation Manager may be used by:

- Financial Management and Planning users working with Performance Management Architect applications
- Financial Management and Planning users working with Classic applications
- Essbase users working with Essbase block storage applications

Oracle Hyperion EPM Architect, Fusion Edition applications work with business rules created in Calculation Manager. Classic Planning, Classic Financial Management, and Essbase block storage applications work with business rules created with Oracle's Hyperion® Business Rules or Calculation Manager. (Runtime prompts are not supported in Essbase rules.)

As in the earlier release, you can launch Financial Management business rulesets and Planning business rules from Oracle Hyperion Financial Management, Fusion Edition and Oracle Hyperion Planning, Fusion Edition, respectively. Oracle Essbase business rules, however, may be launched from either Calculation Manager or Oracle Essbase Administration Services.

Templates Enhancements

Template enhancements include those made to system templates and those made to the Template Designer.

System Templates Enhancements

Hyperion Calculation Manager system templates are now available in wizards instead of in a tab based user interface. This makes the templates easier to use in a business rule and reduces the potential calculation script errors.

The options available in system templates are filtered based on the choices users make when using the wizard. For example, the Aggregation system template does not display the step for selecting the dense dimensions to aggregate if there are no dense dimensions available for aggregation. (This happens if all dense dimensions are used in an upper level member range component.)

System templates can detect upper level member ranges in which they are placed. If a user drops a system template into a member range component (that is, a Fix statement), then the dimensions used in the member range are not displayed in the wizard. This reduces the potential for calculation script syntax errors when using system templates. For example, if the Allocate template is used in a member range made of entities, the entity dimension is not displayed as a dimension on which an allocation can be performed.

Template Designer Enhancements

- Wizard designer

  When you design custom-defined templates, you can define steps in a wizard that guide template users through creating and editing a template and its design time prompts. The templates wizard enables you to organize the display of design time prompts in a template: you can decide what design time prompts you display in each step and choose to display or hide a step based on conditions that you define. Conditions can be based on member or dimension selections made in previous steps or made in an upper Fix statement of the business rule in which the template is used.

- New Upper POV system design time prompt

A new system design time prompt, Upper POV, is displayed in the design time prompt list of each template. This is a member range design time prompt that retrieves the member ranges used in the rule.

- New design time prompt properties

  New properties are available for design time prompts. Design time prompts can be displayed as read only in the wizard. This is useful when you want to display a member selection made in a previous step, but not enable users to edit it.

- New DTP Assignment component for assigning values to design time prompts

  You can assign a value to a design time prompt using the new component, DTP Assignment, that is available for custom-defined templates. In the template flow chart, you can place this component inside a condition component to assign a value to design time prompts based on conditions. Using the DTP Assignment component in a business rule reduces the complexity of the template's flow chart and makes the logic of the template easier to develop and maintain.

  Note: The DTP Assignment component is also used in the design of system templates.

- New condition and condition grid in formula and script components

  For custom-defined templates, formula and script components have a new condition grid that enables you to define a condition for enabling the component. This simplifies the flow chart and prevents you from having to add a separate condition component that can overload the template's flow chart.

  There is also a new condition that lets you test whether a design time prompt type member is dense or sparse.

Printing Enhancements

Printing enhancements:

- You can define the number of pages you want the flow chart to print across and down. Then you can specify whether the components in the flow chart should print down (vertically, like a column), then across (horizontally, like a row) or print across, then down. These options are helpful when you have business rules with many components.
- You can print components in the order in which they display in the Rule Designer or Template Designer flow chart.
- You can insert a page break before a new section (that is, the summary section that includes information about when the rule was created, who owns it, and so on; the variable section that includes information about the variables used in the rule; and the detail section that includes detailed information about the components in the rule).
- You can print nested business rules and rulesets.

Saving Variables Within the Same Application

Using Save As, you can save a variable with a different name to the same application. Before, you could copy a variable from one application to another, but you could not save a variable to the same application.

Zoom Mode Enhancements

When you are in any zoom mode within a flow chart, you can select a component and view and edit its properties. You can also drag a component and drop it into another location in the flow chart in zoom mode. Before, when components in the flow chart were displayed in small sizes, the components were represented by bitmaps that were not selectable.

Formula Grid Enhancements

In the formula grid of a component, you can double-click in a formula's cell to display a larger text box in which to view and edit long formulas.

COPYRIGHT NOTICE

Essbase Addendum, 11.1.1.3

Authors: EPM Information Development Team

ORACLE
ENTERPRISE PERFORMANCE MANAGEMENT SYSTEM