Pro Crystal Enteprise/BusinessObjects XI Programming
by Carl Ganz Jr.

This book was released during the first week of November 2006. Since that time I've compiled a list of items that I felt could be explained better or realized were never explained at all. This addendum contains these further explanations and additions as of June 2007.

Bottom of Page 52

CORRECTION: The book (and the CMS Database Setup user dialog) say that recreating the current data source option will "destroy" the data. This isn't quite true. Only the InfoStore entries will be deleted. Your FileStore will remain as it was, leaving you with a disk full of files and a directory structure that have no corresponding database entries.

Top of page 76

Note: Always load RPT files which are saved without data. BO XI will ignore any data stored with the RPT and you'll just waste disk space.

Bottom of page 87

LifeCycle of a Scheduled Report

When you set up a schedule using the Schedule tab, there are two buttons that need to be understood. The Schedule button actually creates the schedule while the Update button saves the settings as the default so they will remain the next time you enter the Schedule tab. Given that you cannot edit a schedule (the Recurring entry must be deleted and recreated) using the Update button to set defaults makes sense as your settings are already saved and merely need to be edited and rescheduled.

The Recurring entry acts as a baton runner. Rather than initiate another report instance, the scheduled entry kicks off at the appointed time and morphs itself into a Pending, then a Running, and finally a Success or Failed instance. Then, a new Recurring entry is created. This is why the Instance Time of the entry changes each time a scheduled report is run and why you'll never know when the Recurring entry was originally scheduled.

When saving a schedule that only occurs once within a data range - say, selecting every Tuesday between 11/20/2006 (Mon) and 11/25/2006 (Sat) - the Report History entry will display Pending, not Recurring because there is only one Tuesday between these dates. The reason is that the entry is pending until Tuesday and then will never run again before the expiration date of 11/25. Thus, it is not really recurring. The report must run at least twice for the entry to display Recurring. BusinessObjects uses very literal interpretations of these words.

Suppose you create a schedule to run on Tuesday and Wednesday from 11/20 to 11/25 at 1PM. Because the report will run twice between the specified date range, the entry in the Report

History will read Recurring. After it runs for the first time on Tuesday, the entry will change to Pending as it will run only one more time before the end date.

The BeginDate property adjusts itself to reflect the first run date in the schedule. Suppose you create a schedule that runs on Monday and Tuesday between 12/1/2006 (Friday) and 12/31/2006 (Sunday). The BeginDate saved to the InfoStore will be 12/4/2006 as that is the first Monday after 12/1. The EndDate property retains your original date setting of 12/31.

Sometimes when you schedule a report, either programmatically or via the Run Now button, it remains in a Pending state for an extended period, or eternally. One reason this happens is that BO XI is executing too many long running reports simultaneously. By default, BO XI will run five reports in separate processes on one report job server. When one report completes, the next one in the queue will step into the open slot and begin executing. If there is a problem with the stored procedures for some of these reports, say, if it is programmed inefficiently and they either run for hours or run endlessly, these reports never complete and the five slots can fill up. Thus, any report executed subsequently will remain in a Pending state for an extended period. If you see this problem occur, use the Instance Manager to determine which reports have been running for a long time and delete their entries from the InfoStore. Note that even though the entries are deleted the connections to the data source are still in effect and the queries are still running and consuming resources. Thus, additional reports will still go into the Pending state. You can release these resources be killing the Job server or by having your DBA kill the database sessions owned by the jobserverchild process.

Bottom of sidebar on page 116

When checking for a missing custom property your SQL must check against a capitalized NULL as shown in the example. Checking for “= Null” will return zero rows.

Bottom of page 119

The non-indexed properties, which include any custom properties you may create, are stored in a binary format and only the CMS can decipher the contents. Thus, any ideas you may have about indexing them with a tool like Oracle’s InterMedia Text will not be possible.

Bottom of Page 124

Report Instance Date Properties in the InfoStore

Report instances have several date properties that can provide a host of information. Examine the date properties in Table 5-2a. This example is from a report instance that was created from a Recurring instance that was scheduled from Monday-Friday

Table 5-2a InfoStore date properties

Property	Value
SI_INSTANCE	True
SI_UPDATE_TS	5/10/2006 11:25:28 AM

SI_CREATION_TIME	4/14/2006 7:00:39 AM
SI_ENDTIME	4/17/2006 7:02:03 AM
SI_STARTTIME	4/17/2006 7:00:32 AM
SI_NEXTRUNTIME	4/17/2006 7:00:00 AM
MY_SIGN_OFF_DATE	5/10/2006 11:25:28 AM

The SI_CREATION_TIME of this instance - 4/14 - is a Friday. The reason that the SI_STARTTIME is the following Monday is that this is the next scheduled instance of the report that kicked off on 4/17 (Monday). Since the Recurring instance morphed itself into the Friday report and created a new Recurring instance on 4/14, the SI_CREATION_TIME shows 4/14 even though the report did not begin executing until 4/17.

SI_START_TIME shows when the execution of the report began and SI_END_TIME shows when the report completed. The difference between these two times shows how long the report took to execute.

Normally, the SI_UPDATE_TS property, which is indexed, would be equal to or within seconds of the SI_ENDTIME property. In this example, the MY_SIGN_OFF_DATE custom property was added at the date and time indicated to show when the report was signed off by a user. Because the report instance was modified by the addition of this custom property, the SI_UPDATE_TS property was changed accordingly.

Since we're looking at data from a report instance, the SI_NEXTRUNTIME property shows the time the given instance was scheduled to run. You can see that it's very close to the SI_STARTTIME property. The SI_NEXTRUNTIME property for a report instance is of limited use however as instances are never scheduled to run. If you look at the SI_NEXTRUNTIME property for the Recurring instance of this report you'll see that it is scheduled for 4/18/2006 7:00:00 AM (Tuesday).

End of sidebar page 129

The InfoObjects class offers both Count and ResultCount properties. The difference between them becomes apparent when you hit the default 1000 row limit. Suppose you execute a SQL statement that extracts all the instances of a given template and there happen to be 1200 such instances in the InfoStore. Unless you use TOP n in your SQL statement, the InfoObjects object will only contain only 1000 members. Querying the Count property will return 1000 but querying the ResultCount property will return 1200 to reflect the fact that 1200 rows matched the query even though only 1000 rows were returned. If you had used, say a TOP 5000 in your SQL, the InfoObjects object would contain 1200 members and both the Count and ResultCount properties would return 1200.

Middle of page 138

Note: Simply because BO XI shows that a report ran successfully doesn't mean that all is well with the report. The Success flag means that BO XI executed a stored procedure and married the result set to the report template to produce a new report instance. You could find that when you attempt to open this instance that an error triggers because, say, a formula in the report has a problem with the data that was returned. Formulas are only executed when the report is opened so such problems would only manifest themselves then. You should construct a layer of

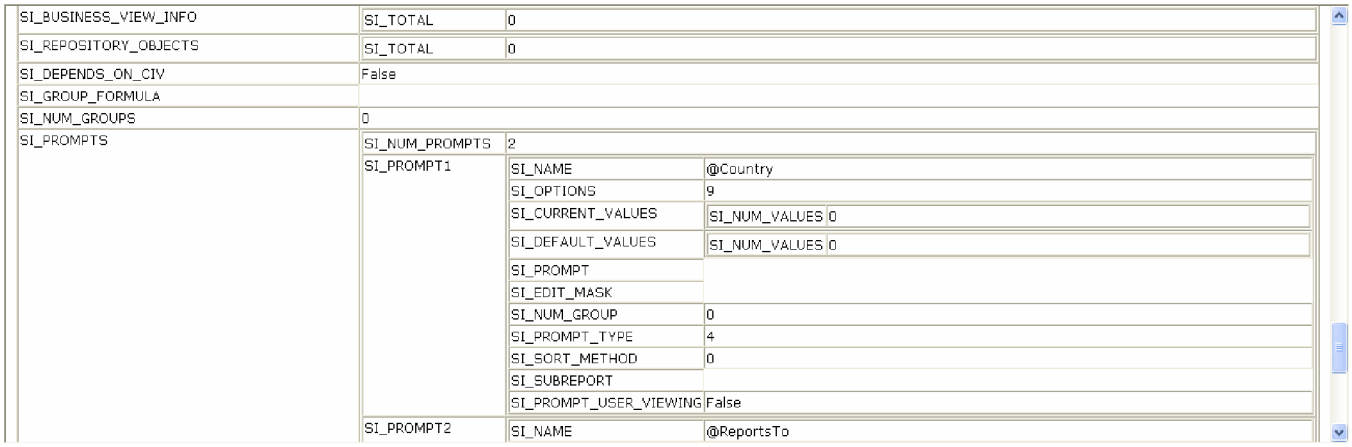
exception handling to any front end that you build that opens report which will handle such contingencies.

Middle of page 139

When you first load a report into the InfoStore, the parameter values are by default set to empty. You can see this by opening the Process|Parameters tab in the CMC. It is good practice to set these empty parameters to a value so you can test the report via the Preview button.

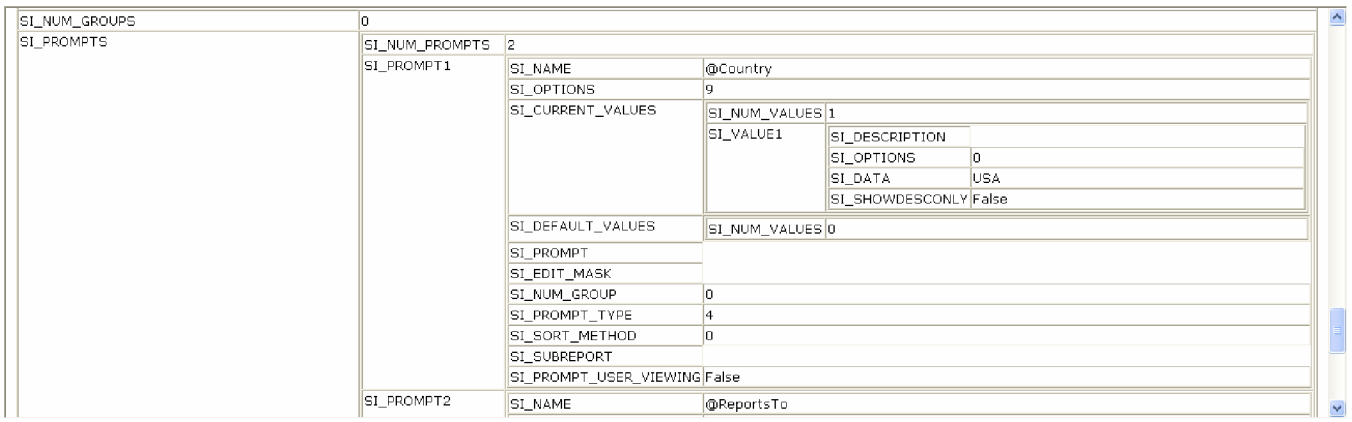
If you are programmatically referencing the parameter values of the report, defaulting these empty parameter values become mandatory. The Query Builder extract in Figure 5.7a shows the prompts for a report that has empty parameters.

Figure 5-7a Empty Paramaters



Note that there is no SI_VALUE1 property under the SI_CURRENT_VALUES property to receive your programmatically passed value. When you enter a parameter value in the CMC this property is automatically created as shown in Figure 5-7b below.

Figure 5-7b Parameter Set



Now that the SI_VALUE1 property exists you can avoid a potential pitfall in your code.

Before code on page 140

Because of the way BO XI stores scheduled entries, you are not always guaranteed to retrieve them all by checking for SI_RECURRING = 1. The section entitled *LifeCycle of a Scheduled Report* found in Chapter 4 explains why. All scheduled entries do have an SI_SCHEDULE_TYPE property however so you could pull all your scheduled entries by filtering for all entries that are not set to run only once as shown here:

```
SELECT *
FROM CI_INFOOBJECTS
WHERE SI_NAME = 'Employee Report'
AND SI_SCHEDULEINFO.SI_SCHEDULE_TYPE != 0
```

The SI_SCHEDULEINFO.SI_SCHEDULE_TYPE property equates to a member of the ceScheduleType enumeration shown in Table 5-5a.

Table 5-5a ceScheduleType enumeration members

Enumerator	Description	Value
ceScheduleType1stMonday	1st Monday of Month	6
ceScheduleTypeCalendar	Calendar	8
ceScheduleTypeCalendarTemplate	Calendar Template	9
ceScheduleTypeDaily	Daily	2
ceScheduleTypeHourly	Hourly	1
ceScheduleTypeLastDay	Last Day of Month	7
ceScheduleTypeMonthly	Monthly	4
ceScheduleTypeNthDay	Nth Day of Month	5
ceScheduleTypeOnce	Once	0
ceScheduleTypeWeekly	Weekly	3

This query will avoid returning entries that are in a transient Pending state waiting to run, as opposed to scheduled entries that are waiting on their last run. A transient Pending state will have a value of zero (ceScheduleTypeOnce). If you need to more accurately restrict the filter, say, by not returning scheduled Pending instances that are not set to run imminently, you can check the SI_NEXTRUNTIME property to see when the Pending entry is set to start. Then, you could create a time buffer by not returning any scheduled reports which are set to run within, say, one hour.

Before Summary section on page 148

Extracting Report History in Columns

The preceding section showed how to extract the report history where the parameters and their values are displayed as a carriage return/line feed delimited string. While display of such data makes for easy reading, a problem arises if you wish to allow the user to sort or search for specific values of the different parameters. For example, suppose you wanted to allow the user to show only those reports which used ‘Smith’ for the last name parameter or which used a BirthDate value that fell within a range of dates. To accomplish this, you’ll need to either store the

parameters as XML text or extract them into separate columns in the table. We'll now examine the latter approach.

The first step is to extract the instance data uses code very similar to that found in Listing 5-29. This code is shown in Listing 5-31.

Listing 5-31 Extracting the prompt information

```
szSQL = "Select SI_ID, SI_NAME, SI_CREATION_TIME, " +
  "SI_SCHEDULE_STATUS, SI_KIND, SI_PROCESSINFO.SI_PROMPTS " +
  "From CI_INFOOBJECTS " +
  "Where SI_PARENTID = " + szParentID +
  " And SI_INSTANCE != 0 " +
  "Order By SI_CREATION_TIME DESC";

oInfoObjects = oInfoStore.Query(szSQL);

oDT = new DataTable();

if (oInfoObjects.Count > 0)
{
  oDT.Columns.Add(new DataColumn("ID"));
  oDT.Columns.Add(new DataColumn("Name"));
  oDT.Columns.Add(new DataColumn("CreationDate"));
  oDT.Columns.Add(new DataColumn("Status"));
  oDT.Columns.Add(new DataColumn("Format"));

  //Build columns that match the prompt names for the most recently run report
  oDT = AddParameterColumns(oInfoObjects[1].ProcessingInfo, oDT);

  foreach (InfoObject oInfoObject in oInfoObjects)
  {
    //Make sure the prompts for this report are the same as for the most recent report.
    //If an older instance has an additional prompt, add it. If it doesn't have a prompt
    //that exists in the most recent instance, the prompt value will remain as null.
    oDT = AdjustParameterColumns(oInfoObject.ProcessingInfo, oDT);

    oDR = oDT.NewRow();

    oDR["ID"] = oInfoObject.Properties["SI_ID"].ToString();
    oDR["Name"] = oInfoObject.Properties["SI_NAME"].ToString();
    oDR["CreationDate"] = oInfoObject.Properties["SI_CREATION_TIME"].ToString();
    oDR["Status"] = GetScheduleStatus(((CeScheduleStatus)
      int.Parse(oInfoObject.Properties["SI_SCHEDULE_STATUS"].ToString())));
    oDR["Format"] = GetFormat(oInfoObject.Properties["SI_KIND"].ToString());

    //Once the basic data has been added, populate the prompt values now that the
    //DataTable structure has been adjusted to receive prompts that existed in older
    //reports but don't exist in the most recent one.
    PopulateParameterColumns(oInfoObject.ProcessingInfo, oDR);

    oDT.Rows.Add(oDR);
  }

}
```

After populating the DataTable with structures for such basic information as creation date and status, we'll use the AddParameterColumns() method to add the structure for the prompts. For example, if the report has a DepartmentID and Salary prompt, the report history DataTable will have columns called DepartmentID and Salary also. The AddParameterColumns() method is shown in Listing 5-32

Listing 5-32 AddParameterColumns() method

```
private DataTable AddParameterColumns(ProcessingInfo oProcessingInfo, DataTable oDT)
{
  Property oProperty;
  int iCnt = 0;
  string szPromptName;

  //How many prompts are there
  iCnt = ((int) oProcessingInfo.Properties["SI_PROMPTS"].Properties["SI_NUM_PROMPTS"].Value);

  if (iCnt != 0)
  {
    //start iterating at the second property
    for (int i=2; i<=iCnt+1; i++)
    {
      //Let's abbreviate our code a bit
      oProperty = oProcessingInfo.Properties["SI_PROMPTS"].Properties[i];

      //Get the name
      szPromptName = ((string) oProperty.Properties["SI_NAME"].Value);

      //Since we're using SQL Server, strip the initial @
      //sign from the prompt to make a valid column name
      szPromptName = szPromptName.Replace("@", string.Empty);

      oDT.Columns.Add(new DataColumn(szPromptName, typeof(string)));
    }
  }

  return oDT;
}
```

In this example, each column is created using a string data type. If you wish, you could query the SI_PROMPT_TYPE property to determine exactly which data type is appropriate.

Once the prompt structure is added, you can then iterate through the InfoObjects that comprise the report history. Because an older report could very easily have a prompt that no longer exists in the current report, the structure of the DataTable would need to be adjusted accordingly. The AdjustParameterColumns() method will check each prompt before it is added to the DataTable to make sure that a column exists to receive it. If it doesn't, the structure is adjusted as needed.

Listing 5-33 AdjustParameterColumns () method

```
private DataTable AdjustParameterColumns(ProcessingInfo oProcessingInfo, DataTable oDT)
{
  Property oProperty;
  int iCnt = 0;
  string szPromptName;

  //How many prompts are there
  iCnt = ((int) oProcessingInfo.Properties["SI_PROMPTS"].Properties["SI_NUM_PROMPTS"].Value);

  if (iCnt != 0)
  {
    //start iterating at the second property
    for (int i=2; i<=iCnt+1; i++)
    {
      //Let's abbreviate our code a bit
      oProperty = oProcessingInfo.Properties["SI_PROMPTS"].Properties[i];
```

```
        //Get the name
        szPromptName = ((string) oProperty.Properties["SI_NAME"].Value);

        //Since we're using SQL Server, strip the initial @
        //sign from the prompt to make a valid column name
        szPromptName = szPromptName.Replace("@", string.Empty);

        //Since this older report instance could contain a prompt that doesn't exist
        //in the most recent instance, see if it's in the current DataTable structure.
        //If not, add it.
        if (! IsColumnExists(oDT.Columns, szPromptName))
          oDT.Columns.Add(new DataColumn(szPromptName, typeof(string)));

      }
    }

    return oDT;
}
```

This structure check must be performed for each entry in the report history. Once the structure is adjusted, the parameter values may then be added as show in Listing 5-34

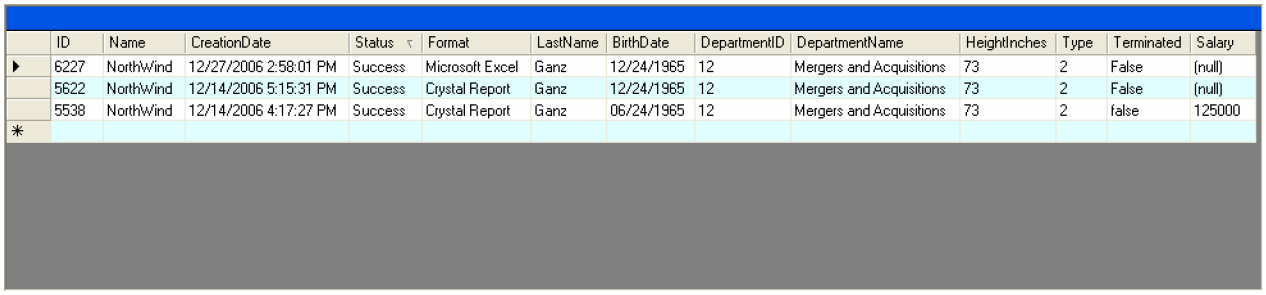
Listing 5-34 PopulateParameterColumns() method

```
private DataRow PopulateParameterColumns(ProcessingInfo oProcessingInfo, DataRow oDR)
{
    Property oProperty;
    int iCnt = 0;
    string szPromptName;
    string szPromptValue;

    //How many prompts are there
    iCnt = ((int) oProcessingInfo.Properties["SI_PROMPTS"].Properties["SI_NUM_PROMPTS"].Value);

    if (iCnt != 0)
    {
      //start iterating at the second property
      for (int i=2; i<=iCnt+1; i++)
      {
        //Let's abbreviate our code a bit
        oProperty = oProcessingInfo.Properties["SI_PROMPTS"].Properties[i];

        //Get the name
        szPromptName = ((string) oProperty.Properties["SI_NAME"].Value);

        szPromptName = szPromptName.Replace("@", string.Empty);

        try
        {
          szPromptValue = ((string) oProperty.Properties["SI_CURRENT_VALUES"].
            Properties["SI_VALUE1"].Properties["SI_DATA"].Value);

          szPromptValue = FixDate(szPromptValue);
        }
        catch
        {
          szPromptValue = string.Empty;
        }

        oDR[szPromptName] = szPromptValue;
      }
    }
```

```
    return oDR;
}
```

This code will add the individual prompt values to the DataTable. The FixDate() method, available in the code download, converts date parameters which are stored as “DateTime(yyyy, mm, dd, hh, mm, ss)” to a “mm/dd/yyyy” format. The result is the display shown in Figure 5-9. The LastName, BirthDate, DepartmentID, DepartmentName, HeightInches, Type, Terminated, and Salary columns represent the report parameters. The Salary parameter existed on older reports but was later removed. Thus, the Salary column contains a null value for the most recent reports.

Figure 5-9 Parameter values as separate columns



	ID	Name	CreationDate	Status	Format	LastName	BirthDate	DepartmentID	DepartmentName	HeightInches	Type	Terminated	Salary
▶	6227	NorthWind	12/27/2006 2:58:01 PM	Success	Microsoft Excel	Ganz	12/24/1965	12	Mergers and Acquisitions	73	2	False	(null)
	5622	NorthWind	12/14/2006 5:15:31 PM	Success	Crystal Report	Ganz	12/24/1965	12	Mergers and Acquisitions	73	2	False	(null)
	5538	NorthWind	12/14/2006 4:17:27 PM	Success	Crystal Report	Ganz	06/24/1965	12	Mergers and Acquisitions	73	2	false	125000
*													

Top of page 150

Note: Whenever you add email addresses using the object model, be sure to add them individually using the Add method. You cannot add email addresses by applying a delimited string of multiple addresses. If you do so they will all be treated as one long address and you will receive an SMTP address error.

Top of page 152

Note: When you schedule a report and create a Recurring instance, that instance copies to itself the notification information that was in effect when the schedule was created. If you change the notification information in any way you’ll need to delete the schedule and recreate it for the new settings to take effect.

Bottom of page 176

Calendars can be associated with schedules so your reports can recur in a very customized fashion. Unlike success/failure notifications, the recurring instance does not absorb the calendar settings when the schedule is first created. Doing so would preclude the recurring instance from recognizing any subsequent changes to the calendar. Rather, the recurring instance maintains only a referential link to the calendar objects using the SI_CALENDAR_TEMPLATE_ID property.

Any changes to the calendar are immediately recognized without needing to drop and recreate the schedule. Should you delete a calendar that is used in a schedule you'll receive no error or warning from either the SDK or the CMC. You need to first check for use of the specific calendar object via the SQL statement below:

```
SELECT SI_AGGREGATE_COUNT
FROM CI_INFOOBJECTS
WHERE SI_SCHEDULEINFO.SI_CALENDAR_TEMPLATE_ID = <CalendarID>
```

Bottom of page 192

The Office documents you stream to your browser may open directly in Internet Explorer instead of in the appropriate Office program. When they do so you will not have the toolbar that allows you to save, etc. This behavior may occur if IE is configured to host documents for Office programs that are installed on the local workstation. By default, Internet Explorer is configured to do just this. To resolve this issue, take the following steps:

1. Open My Computer.
2. On the Tools menu (or the View menu), click Folder Options (or click Options).
3. Click the File Types tab.
4. In the Registered file types list, click the specific Office document type (for example, Microsoft Excel Worksheet), and then click Advanced (or click Edit).
5. In the Edit File Type dialog box, click to clear the Browse in same window check box (or click to clear the Open Web documents in place check box).
6. Click OK.

You can learn more about this issue by reading Microsoft KnowledgeBase article 162059.

Page 231

In the sections that follow you'll see a number of references to objects with "Controller" as part of their names. The RAS manages reports via a series of these Controller objects as detailed in Table 8-0 below:

Table 8-0 Controller Objects

CustomFunctionController
DatabaseController
DataDefController
FilterController
FormulaFieldController
GroupController
ParameterFieldController
PrintOutputController
ReportAreaController
ReportController
ReportSectionController
ResultFieldController
RowsetController
SearchController
SortController
SubreportController

SummaryFieldController

Top of page 272

Accessing Data within a Report

Using the RAS you can access the data that is stored within a report. There are a series of RowSet classes which allow you to accomplish this. Suppose we want to extract the data from a report and store it in a DataTable. The code in Listing 8-36a shows you how this is done.

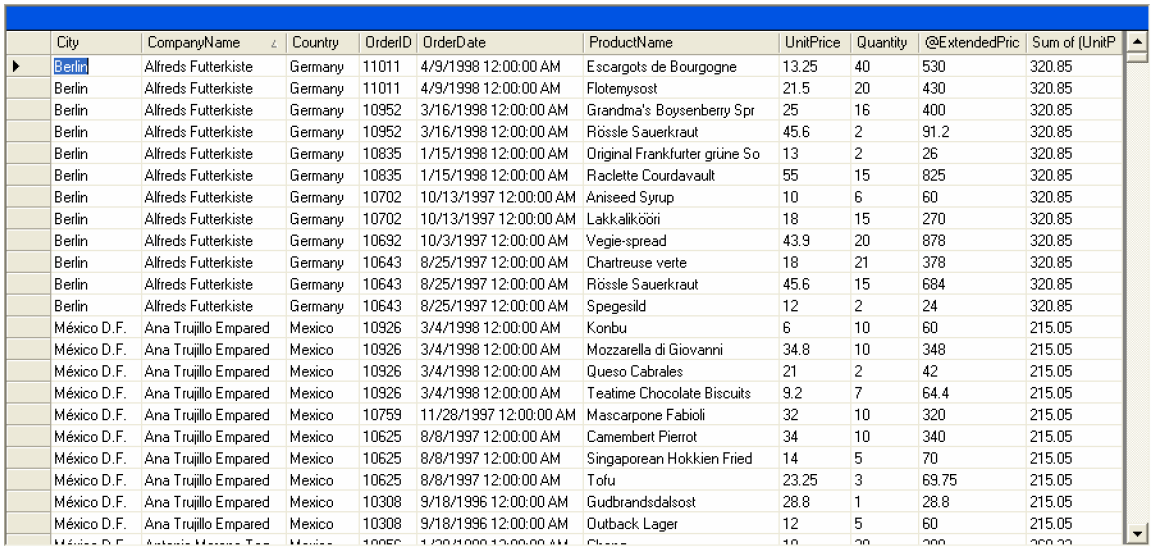
Listing 8-36a Accessing data within a report

```
ReportClientDocumentClass oReportClientDocument;
RowsetMetaData oRowsetMetaData;
RowsetController oRowsetController;
RowsetCursor oRowsetCursor;
Record oRecord;
DataTable oDT;
DataRow oDR;
object oTarget;
object oTargetXML;
string szFieldName;
string szSQL;
int iRow = 0;
int iRecs = 0;
int iCnt;

oTarget = @"C:\temp\testreport.rpt";

//Open the Rpt file
oReportClientDocument = new ReportClientDocumentClass();
oReportClientDocument.Open(ref(oTarget), 1);

//Instantiate RowsetMetaData class and initialize it with a set of
//data fields that is equal to the structure of the report
oRowsetMetaData = new RowsetMetaDataClass();
oRowsetMetaData.DataFields = oReportClientDocument.DataDefController.DataDefinition.ResultFields;

//Extract the RowSetController object which provides access to the underlying data. Setting the
//RowsetBatchSize property to -1 indicates that all records should be returned.
oRowsetController = oReportClientDocument.RowsetController;
oRowsetController.RowsetBatchSize = -1;

//Create a DataTable object...
oDT = new DataTable();

//...and iterate through each field in the report's structure to create DataColumns
//objects of the same names
foreach (Field oField in oRowsetController.DataDefinition.ResultFields)
{
  szFieldName = oField.get_DisplayName(CrFieldDisplayNameTypeEnum.crFieldDisplayNameDescription,
    CrystalDecisions.ReportAppServer.DataDefModel.CeLocale.ceLocaleUserDefault);

  oDT.Columns.Add(new DataColumn(szFieldName, typeof(string)));
}
```

```
//Create a RowsetCursor object and move the record pointer to the first row
oRowsetCursor = oRowsetController.CreateCursor(null, oRowsetMetaData, 0);
oRowsetCursor.MoveTo(iRow, 0);

//Determine the number of records found in the report
iRecs = oRowsetCursor.RecordCount;

//...and the number of columns in the DataTable
iCnt = oDT.Columns.Count;

//Iterate through the RowSetCursor Object
while(! oRowsetCursor.IsEOF && oRowsetCursor.CurrentRecordNumber < iRecs)
{
  //Extract a reference to the current record
  oRecord = oRowsetCursor.CurrentRecord;

  oDR = oDT.NewRow();

  //Iterate through the fields in the record and add the data
  //to the matching fields in the DataTable
  for(int i=0; i<iCnt; i++)
  {
    //Check here for missing data. For example, one of the Northwind
    //customers has no orders so while there exists a customer name,
    //city, and country there is no order information. Not checking for a null
    //value would cause an error in this case
    if (oRecord[i] != null)
      oDR[i] = oRecord[i].ToString();
  }
  oDT.Rows.Add(oDR);

  oRowsetCursor.MoveNext();
}

dataGrid1.DataSource = oDT;
```

This code produces the output shown in Figure 8-10a and matches the report shown in Figure 8-10.

Figure 8-10a Denormalized output from NorthWind report



City	CompanyName	Country	OrderID	OrderDate	ProductName	UnitPrice	Quantity	@ExtendedPric	Sum of (UnitP
Berlin	Alfreds Futterkiste	Germany	11011	4/9/1998 12:00:00 AM	Escargots de Bourgogne	13.25	40	530	320.85
Berlin	Alfreds Futterkiste	Germany	11011	4/9/1998 12:00:00 AM	Flotemysost	21.5	20	430	320.85
Berlin	Alfreds Futterkiste	Germany	10952	3/16/1998 12:00:00 AM	Grandma's Boysenberry Spr	25	16	400	320.85
Berlin	Alfreds Futterkiste	Germany	10952	3/16/1998 12:00:00 AM	Rössle Sauerkraut	45.6	2	91.2	320.85
Berlin	Alfreds Futterkiste	Germany	10835	1/15/1998 12:00:00 AM	Original Frankfurter grüne So	13	2	26	320.85
Berlin	Alfreds Futterkiste	Germany	10835	1/15/1998 12:00:00 AM	Raclette Courdavault	55	15	825	320.85
Berlin	Alfreds Futterkiste	Germany	10702	10/13/1997 12:00:00 AM	Aniseed Syrup	10	6	60	320.85
Berlin	Alfreds Futterkiste	Germany	10702	10/13/1997 12:00:00 AM	Lakkalikööri	18	15	270	320.85
Berlin	Alfreds Futterkiste	Germany	10692	10/3/1997 12:00:00 AM	Vegie-spread	43.9	20	878	320.85
Berlin	Alfreds Futterkiste	Germany	10643	8/25/1997 12:00:00 AM	Chartreuse verte	18	21	378	320.85
Berlin	Alfreds Futterkiste	Germany	10643	8/25/1997 12:00:00 AM	Rössle Sauerkraut	45.6	15	684	320.85
Berlin	Alfreds Futterkiste	Germany	10643	8/25/1997 12:00:00 AM	Spegesild	12	2	24	320.85
México D.F.	Ana Trujillo Emparedad	Mexico	10926	3/4/1998 12:00:00 AM	Konbu	6	10	60	215.05
México D.F.	Ana Trujillo Emparedad	Mexico	10926	3/4/1998 12:00:00 AM	Mozzarella di Giovanni	34.8	10	348	215.05
México D.F.	Ana Trujillo Emparedad	Mexico	10926	3/4/1998 12:00:00 AM	Queso Cabrales	21	2	42	215.05
México D.F.	Ana Trujillo Emparedad	Mexico	10926	3/4/1998 12:00:00 AM	Teatime Chocolate Biscuits	9.2	7	64.4	215.05
México D.F.	Ana Trujillo Emparedad	Mexico	10759	11/28/1997 12:00:00 AM	Mascarpone Fabioli	32	10	320	215.05
México D.F.	Ana Trujillo Emparedad	Mexico	10625	8/8/1997 12:00:00 AM	Camembert Pierrot	34	10	340	215.05
México D.F.	Ana Trujillo Emparedad	Mexico	10625	8/8/1997 12:00:00 AM	Singaporean Hokkien Fried	14	5	70	215.05
México D.F.	Ana Trujillo Emparedad	Mexico	10625	8/8/1997 12:00:00 AM	Tofu	23.25	3	69.75	215.05
México D.F.	Ana Trujillo Emparedad	Mexico	10308	9/18/1996 12:00:00 AM	Gudbrandsdalsost	28.8	1	28.8	215.05
México D.F.	Ana Trujillo Emparedad	Mexico	10308	9/18/1996 12:00:00 AM	Outback Lager	12	5	60	215.05

It’s important to understand that the RowsetCursor can only access the database fields in the report. Text fields, running totals, and formulas are not available to you through these objects. If it is important to retrieve formulas you could perform all your calculations in the stored procedure and then display the results as database fields which can be read.

The RowsetController object will also export the reports data to XML as show here:

```
oTargetXML = @"C:\temp\testreport.xml";

oRowsetController.SaveToXML(oTargetXML, oRowsetCursor.GroupPath,
   oRowsetCursor.Rowset, 0, 0, -1, -1, true);
```

You’ll need to indicate the target file name, the GroupPath of the RowsetCursor, the Rowset object, and the starting row and column which in this example are both zero. You’ll also need to indicate the number of rows and columns to include. Specifying -1 for both indicate all rows and columns. Finally, indicate if special fields should be included. This method will produce the output shown in Listing 8-36b.

Listing 8-36b XML output

```
  <xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882" xmlns:dt="uuid:C2F41010-65B3-11d1-
A29F-00AA00C14882" xmlns:rs="urn:schemas-microsoft-com:rowset" xmlns:z="#RowsetSchema">
  <s:Schema ID="RowsetSchema">
  <s:ElementType name="row">
  <s:AttributeType name="C0" rs:name="spc_SalesOrders;1.City" rs:nullable="true">
  <s:datatype dt:type="string" dt:maxLength="32" />
  </s:AttributeType>
  <s:AttributeType name="C1" rs:name="spc_SalesOrders;1.CompanyName" rs:nullable="true">
  <s:datatype dt:type="string" dt:maxLength="82" />
  </s:AttributeType>
  <s:AttributeType name="C2" rs:name="spc_SalesOrders;1.Country" rs:nullable="true">
  <s:datatype dt:type="string" dt:maxLength="32" />
  </s:AttributeType>
  <s:AttributeType name="C3" rs:name="spc_SalesOrders;1.OrderID" rs:nullable="true">
  <s:datatype dt:type="int" dt:maxLength="4" />
  </s:AttributeType>
  <s:AttributeType name="C4" rs:name="spc_SalesOrders;1.OrderDate" rs:nullable="true">
  <s:datatype dt:type="dateTime" dt:maxLength="8" />
  </s:AttributeType>
  <s:AttributeType name="C5" rs:name="spc_SalesOrders;1.ProductName" rs:nullable="true">
  <s:datatype dt:type="string" dt:maxLength="82" />
  </s:AttributeType>
  <s:AttributeType name="C6" rs:name="spc_SalesOrders;1.UnitPrice" rs:nullable="true">
  <s:datatype dt:type="number" dt:maxLength="8" />
  </s:AttributeType>
  <s:AttributeType name="C7" rs:name="spc_SalesOrders;1.Quantity" rs:nullable="true">
  <s:datatype dt:type="int" dt:maxLength="4" />
  </s:AttributeType>
  <s:AttributeType name="C8" rs:name="@ExtendedPrice" rs:nullable="true">
  <s:datatype dt:type="number" dt:maxLength="8" />
  </s:AttributeType>
  <s:AttributeType name="C9" rs:name="Sum (spc_SalesOrders;1.UnitPrice,
spc_SalesOrders;1.CompanyName)" rs:nullable="true">
  <s:datatype dt:type="number" dt:maxLength="8" />
  </s:AttributeType>
  <s:AttributeType name="C10" rs:name="RecordNumber" rs:nullable="true">
  <s:datatype dt:type="ui4" dt:maxLength="4" />
  </s:AttributeType>
  </s:ElementType>
  </s:Schema>
```

```
  <rs:data>
  <z:row C0="Berlin" C1="Alfreds Futterkiste" C2="Germany" C3="11011" C4="1998-4-9T0:0:0.0"
C5="Escargots de Bourgogne" C6="13.25" C7="40" C8="530" C9="320.85" C10="0" __FirstRecNum_="0"
/>
  <z:row C0="Berlin" C1="Alfreds Futterkiste" C2="Germany" C3="11011" C4="1998-4-9T0:0:0.0"
C5="Flotemysost" C6="21.5" C7="20" C8="430" C9="320.85" C10="1" />
  <z:row C0="Berlin" C1="Alfreds Futterkiste" C2="Germany" C3="10952" C4="1998-3-16T0:0:0.0"
C5="Grandma's Boysenberry Spread" C6="25" C7="16" C8="400" C9="320.85" C10="2" />
```

Exporting Report Data to Excel/HTML

The Crystal Reports engine doesn't always handle the export to Excel nicely. Excel also makes decisions about data types that aren't always correct. BusinessObjects publishes a white paper that discusses how to format your report to reap the maximum benefit when exporting to Excel. Between the two, you may very well end up with less than optimal data formatting in your spreadsheet output.

One easy way around this is to customize the Excel output yourself. Using the RowsetCursor and RowsetController objects you can create your own Excel export by converting the report's data to an HTML table. Examine the code in Listing 8-36c.

Listing 8-36c Creating an HTML header

```
iRecs = oRowsetCursor.RecordCount;

iCnt = oRowsetController.DataDefinition.ResultFields.Count;

dataString = new StringBuilder();

//Start the HTML table and begin the header row
dataString.Append("<TABLE border='1'>");
dataString.Append("<TR>");

//Iterate through the fields collection to create the column headers
foreach (Field oField in oRowsetController.DataDefinition.ResultFields)
{
  szFieldName = oField.get_DisplayName(CrFieldDisplayNameTypeEnum.crFieldDisplayNameDescription,
    CrystalDecisions.ReportAppServer.DataDefModel.CeLocale.ceLocaleUserDefault);

  dataString.Append("<TH>");
  dataString.Append(szFieldName);
  dataString.Append("</TH>");
}

//Close the header row
dataString.Append("</TR>");
```

Here, the DataDefinition object of the RowsetController is used to supply the list of field names to create the header. Each field name is delimited by a <TH> tag. These column headers will appear bold in the spreasdsheet. Then, the code in Listing 8-36d will export the body of the report.

Listing 8-36d Exporting the body of the report

```
//Iterate through the RowsetCursor object
while(! oRowsetCursor.IsEOF && oRowsetCursor.CurrentRecordNumber < iRecs)
{
  oRecord = oRowsetCursor.CurrentRecord;
```

```
  //Begin a data row
  dataString.Append("<TR>");

  //Iterate through each fields in the record
  for(int i=0; i<iCnt; i++)
  {
    if (oRecord[i] != null)
      szData = oRecord[i].ToString();
    else
      szData = string.Empty;

    szAlign = "left";

    sCrFieldValueTypeEnum = oRowsetController.DataDefinition.ResultFields[i].Type;

    //If the field type is a date or date/time, format as dd-MMM-yyyy
    if (sCrFieldValueTypeEnum == CrFieldValueTypeEnum.crFieldValueTypeDateField ||
      sCrFieldValueTypeEnum == CrFieldValueTypeEnum.crFieldValueTypeDateTimeField)
    {
      if (IsDate(szData))
          szData = DateTime.Parse(szData).ToString("dd-MMM-yyyy");
    }

    //If dealing with a numeric field make sure to right-justify
    if (sCrFieldValueTypeEnum == CrFieldValueTypeEnum.crFieldValueTypeCurrencyField ||
      sCrFieldValueTypeEnum == CrFieldValueTypeEnum.crFieldValueTypeDecimalField ||
      sCrFieldValueTypeEnum == CrFieldValueTypeEnum.crFieldValueTypeNumberField)
      szAlign = "right";

    dataString.Append("<TD align=" + szAlign + ">");
    dataString.Append(szData);
    dataString.Append("</TD>");
  }

  dataString.Append("</TR>");

  oRowsetCursor.MoveNext();
}

//Close the table
dataString.Append("</TABLE>");
```

In this code the individual data elements are exported row by row and element by element. If a field's data type is a date or date/time then the data value is formatted to dd-MMM-yyyy to provide an internationally recognized date standard, for example "12-Jan-2007". All numeric data types are left aligned. You'll likely need to tweak this code to customize it for your specific purposes but it offers most of what you need to get started.

Searching Report Data

Though it's certainly nice to have the facility to extract the data from a report, it's even nicer to be able to filter that data. The code in Listing 8-36e opens an RPT file and sets a filter restricting the output to orders for a specific company. Then, the output is written to an XML file. Though this example returns the data as a SearchResultCursor object, you can still iterate through this object and perform such as actions as writing the data to a DataTable as shown in the preceding section.

Listing 8-36e Filtering report data

```
SearchController oSearchController;
SearchResultCursor oSearchResultCursor;
Filter oFilter;
ReportClientDocumentClass oReportClientDocument;
RowsetMetaData oRowsetMetaData;
object oTarget;
object oTargetXML;

oTarget = @"C:\temp\testreport.rpt";

//Open the Rpt file
oReportClientDocument = new ReportClientDocumentClass();
oReportClientDocument.Open(ref(oTarget), 1);

//Instantiate RowsetMetaData class and initialize it with a set of
//data fields that is equal to the structure of the report
oRowsetMetaData = new RowsetMetaDataClass();
oRowsetMetaData.DataFields = oReportClientDocument.DataDefController.DataDefinition.ResultFields;

//Instantiate the SearchController object
oSearchController = oReportClientDocument.SearchController;

//Create a Filter object
oFilter = new Filter();
oFilter.FreeEditingText = "{spc_SalesOrders;1.CompanyName} = 'Around the Horn'";

//Create a cursor based on the filter object
oSearchResultCursor = oSearchController.CreateCursor(null, oRowsetMetaData, oFilter, true, 0);

//Move to the first row...
oSearchResultCursor.MoveTo(0, 0);

//...and send the filtered output to an XML file.
oTargetXML = @"C:\temp\FilteredOutput.xml";

oSearchResultCursor.Rowset.SaveToXML(oTargetXML);
```

In this example, the filter is provided by creating a Filter object and setting the FreeEditingText property. You could achieve the same result by using an object of the FieldRangeFilterItemClass class. To do this, you'll need to specify values for the RangeField property, the Operations property, and the Values collection. The RangeField property receives a Field object as shown in Listing 8-36f.

Listing 8-36f Filtering report data

```
//Find the table or stored procedure in the table collection based on the name
oISCRTable = oReportClientDocument.Database.Tables.FindTableByAlias("spc_SalesOrders;1");

//Extract the table or stored procedure and cast it to a Table object
oTable = ((Table) oISCRTable);

//Cast this field reference to a Field object
oField = ((Field) oTable.DataFields.FindField("UnitPrice",
  CrFieldDisplayNameTypeEnum.crFieldDisplayNameName,
  CrystalDecisions.ReportAppServer.DataDefModel.CeLocale.ceLocaleUserDefault));

oFieldRangeFilterItem.RangeField = oField;
```

Next, you can set the Operations property to one of the members of the CrSelectionOperationEnum enumerator. In this example

oFieldRangeFilterItem.Operation = CrSelectionOperationEnum. crSelectionOperationGreaterThan;

Finally, you can populate the Values property like this:

oFieldRangeFilterItem.Values.Add(100);

Listing 8-36g shows a few examples of how to use the FieldRangeFilterItemClass class.

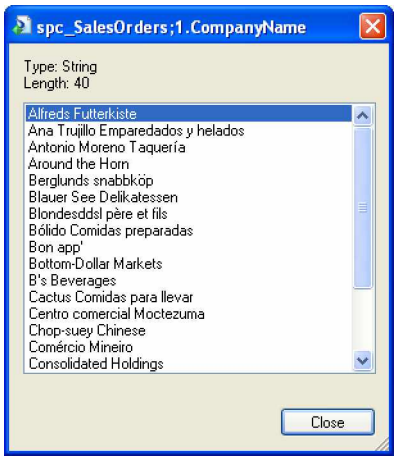
Listing 8-36g Using the FieldRangeFilterItemClass class

```
oFieldRangeFilterItem = new FieldRangeFilterItemClass();

//Searching for a specific company name
oFieldRangeFilterItem.RangeField = oField;
oFieldRangeFilterItem.Operation = CrSelectionOperationEnum.crSelectionOperationEqual;
oFieldRangeFilterItem.Values.Add("Around the Horn");

//Restricting output to a list of company names
oFieldRangeFilterItem.RangeField = oField;
oFieldRangeFilterItem.Operation = CrSelectionOperationEnum.crSelectionOperationOneOf;
oFieldRangeFilterItem.Values.Add("Around the Horn");
oFieldRangeFilterItem.Values.Add("Island Trading");

//Looking for a field value, for example UnitPrice, greater than 100
oFieldRangeFilterItem.RangeField = oField;
oFieldRangeFilterItem.Operation = CrSelectionOperationEnum.crSelectionOperationGreaterThan;
oFieldRangeFilterItem.Values.Add(100);

oFilter.FilterItems.Add(oFieldRangeFilterItem);
```

BrowseFieldValues and BrowseTableValues methods

The BrowseTableValues and BrowseTableValues methods allow you to provide a preview of the data in your report. Crystal Reports has similar functionality. If you right click on a field in the File d Explorer you'll see a menu item labeled Browse data. Choosing this item will display a dialog box similar to the one shown in Figure 8-10b.

Figure 8-10b Browse Data dialog box



You can create a dialog box of your own using the BrowseFieldValues and BrowseTableValues methods of the RowsetController object. These methods take Field and Table objects, respectively, and a second parameter indicating how many records to return. The code in Listing 8-36h shows how to extract the first ten rows of data for the Country column of the report's data source.

Listing 8-36h BrowseFieldValues method in action

```
//Find the table or stored procedure in the table collection based on the name
oISCRTable = oReportClientDocument.Database.Tables.FindTableByAlias("spc_SalesOrders;1");

//Extract the table or stored procedure and cast it to a Table object
oTable = ((Table) oISCRTable);

//Cast this field reference to a Field object
oField = ((Field) oTable.DataFields.FindField("Country",
  CrFieldDisplayNameTypeEnum.crFieldDisplayNameName,
  CrystalDecisions.ReportAppServer.DataDefModel.CeLocale.ceLocaleUserDefault));

oValues = oRowsetController.BrowseFieldValues(oField, 10);

oDT = new DataTable();

oDT.Columns.Add(new DataColumn("Data"));

foreach(Value oValue in oValues)
{
  oDR = oDT.NewRow();

  oDR["Data"] =oValue.DisplayText(CrystalDecisions.ReportAppServer.
    DataDefModel.CeLocale.ceLocaleUserDefault);

  oDT.Rows.Add(oDR);
}

dataGrid1.DataSource = oDT;
```

The BrowseFieldValues method returns a Values collection objects which contains discrete data elements containing the data contained within that requested field. Note that the data returned is not filtered for unique values as the Crystal Reports dialog box is. To achieve this you'll need to modify the code to add only unique values to the DataTable.

Similarly, you can view the first ten rows of report data using the code shown in Listing 8-36i. Here, the BrowseTableValues method returns a Records object collection, comprised of individual Record objects. By creating a DataTable of the same structure as the Table object, you can collect the contents of the Records collection and display the results in a DataGrid.

Listing 8-36i BrowseTableValues method in action

```
//Find the table or stored procedure in the table collection based on the name
oISCRTable = oReportClientDocument.Database.Tables.FindTableByAlias("spc_SalesOrders;1");

//Extract the table or stored procedure and cast it to a Table object
oTable = ((Table) oISCRTable);

oRecords = oRowsetController.BrowseTableValues(oTable, 10);

//Create a DataTable object...
oDT = new DataTable();
```

```
//...and iterate through each field in the report's structure to create DataColumns
//objects of the same names
foreach (Field oField in oTable.DataFields)
{
  szFieldName = oField.get_DisplayName(CrFieldDisplayNameTypeEnum.
    crFieldDisplayNameDescription, CrystalDecisions.ReportAppServer.
    DataDefModel.CeLocale.ceLocaleUserDefault);

  oDT.Columns.Add(new DataColumn(szFieldName, typeof(string)));
}

iCnt = oDT.Columns.Count;

//Iterate through the fields in the record and add the data
//to the matching fields in the DataTable
foreach (Record oRecord in oRecords)
{
  //Check here for missing data. For example, one of the Northwind
  //customers has no orders so while there exists a customer name,
  //city, and country there is no order information. Not checking for a null
  //value would cause an error in this case
  if (oRecord != null)
  {
    oDR = oDT.NewRow();

    for (int i=0; i<iCnt; i++)
      oDR[i] = oRecord[i].ToString();

    oDT.Rows.Add(oDR);
  }
}

dataGrid1.DataSource = oDT;
```

Figure 8-10c shows the output of this code.

Figure 8-10c Results of BrowseTableValues method

CustomerI	CompanyName	City	Country	OrderID	OrderDate	Employee	ProductName
ALFKI	Alfreds Futterkiste	Berlin	Germany	10643	8/25/1997 12:00:00 AM	Michael Suyama	Chartreuse verte
ALFKI	Alfreds Futterkiste	Berlin	Germany	10643	8/25/1997 12:00:00 AM	Michael Suyama	Rössle Sauerkraut
ALFKI	Alfreds Futterkiste	Berlin	Germany	10643	8/25/1997 12:00:00 AM	Michael Suyama	Spegesild
ALFKI	Alfreds Futterkiste	Berlin	Germany	10692	10/3/1997 12:00:00 AM	Margaret Peacock	Vegie-spread
ALFKI	Alfreds Futterkiste	Berlin	Germany	10702	10/13/1997 12:00:00 AM	Margaret Peacock	Aniseed Syrup
ALFKI	Alfreds Futterkiste	Berlin	Germany	10702	10/13/1997 12:00:00 AM	Margaret Peacock	Lakkalikööri
ALFKI	Alfreds Futterkiste	Berlin	Germany	10835	1/15/1998 12:00:00 AM	Nancy Davolio	Original Frankfurter grüne Soße
ALFKI	Alfreds Futterkiste	Berlin	Germany	10835	1/15/1998 12:00:00 AM	Nancy Davolio	Raclette Courdavault
ALFKI	Alfreds Futterkiste	Berlin	Germany	10952	3/16/1998 12:00:00 AM	Nancy Davolio	Grandma's Boysenberry Spread
ALFKI	Alfreds Futterkiste	Berlin	Germany	10952	3/16/1998 12:00:00 AM	Nancy Davolio	Rössle Sauerkraut

Bottom of page 289

Handling Long-running Reports

Though an on-demand web service is a powerful tool, you'll need to consider the possibility that some reports will take more time to run that a user is willing to wait. If the report takes more than a few seconds to complete, you don't want to tie up the application such that the user cannot

move on to other tasks. Rather, you can run the report on a separate thread and send it to the user when it is complete. Though threading can be a dangerous area of programming if you're unfamiliar with it, running reports in threads requires only a basic knowledge of threading and does not require you to trek into unsafe territory. Using threads, the application will return control of the application to the user immediately. When the report is complete, it will display the report immediately to the user.

Because you don't know what the user will be doing at the time, displaying the report immediately upon completion may not be feasible. If your user is in the middle of data entry on a screen, he'll likely not appreciate a report popping up while he's typing. The code in Listing 9.x illustrates how to send the report to SMTP mail so the user will receive the report via email when its done. Listing 9.11a shows how to accomplish this.

Listing 9.11a Sending the report to the user via email

```
//set the report object's format options
oReportFormatOpts = oReportPlugIn.ReportFormatOptions;
oReportFormatOpts.Format = sReportFormat;

//Create an interface to the scheduling options for the report.
oSchedulingInfo = oReport.SchedulingInfo;

//run the report right now
oSchedulingInfo.RightNow = true;

//run the report once only
oSchedulingInfo.Type = CeScheduleType.ceScheduleTypeOnce;

//Pull the Smtp object instead of the DiskUnmanaged object
szSQL = "Select Top 1* " +
            "From CI_SYSTEMOBJECTS " +
            "Where SI_PARENTID = 29 " +
            "And SI_NAME = 'CrystalEnterprise.Smtp'";

oDestinationObjects = oInfoStore.Query(szSQL);

oDestinationObject = oDestinationObjects[1];

//Create the DestinationPlugin object
oDestinationPlugin = new DestinationPlugin(oDestinationObjects[1].PluginInterface);

oSmtpOptions = new SmtpOptions(oDestinationPlugin.ScheduleOptions);

oSmtpOptions.EnableAttachments = true;
oSmtpOptions.SMTPAuthentication = CeAuthentication.ceAuthLogin;
oSmtpOptions.SMTPPassword = "mypassword";
oSmtpOptions.SMTPUserName = "myuserName";
oSmtpOptions.ServerName = "mysmtpserver";
oSmtpOptions.DomainName = "mydomain";
oSmtpOptions.Port = 80;
oSmtpOptions.SenderAddress = "BOXIAdmin@mycompany.com";
oSmtpOptions.ToAddresses.Add(szEmail);
oSmtpOptions.Delimiter = ";";
oSmtpOptions.Subject = "Daily Sales Report";
oSmtpOptions.Message = "Click here %SI_VIEWER_URL% to view your report.";

oDestination = oReport.SchedulingInfo.Destination;

oDestination.SetFromPlugin(oDestinationPlugin);
```

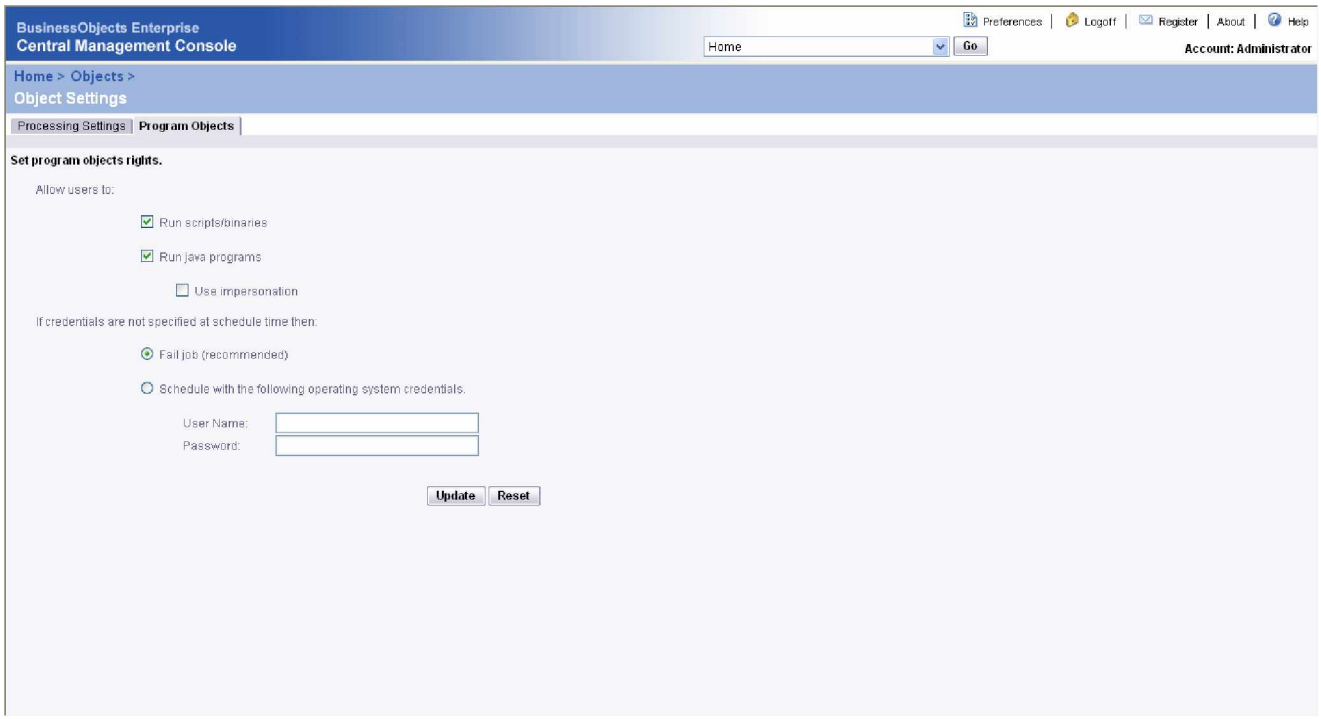
oInfoStore.Schedule(oReports);

Passing the email address to the web service will indicate that the report should be transmitted in this fashion. If multiple recipients are involved, pass a delimited list of email addresses, parse them, and add them individually using the Add method. The Delimiter property of the SmtpOptions object indicates what this delimiter is and the object will parse accordingly.

Middle of page 290

Before you can run EXEs you'll need to enable them to run. You can do this by selecting the Object Settings button on the Objects page. Objects Settings will offer you two tabs. Select the Program Objects tab and you'll see the screen shown in Figure 9-0.

Figure 9-0 Program Objects page



Click the Run Scripts/Binaries checkbox to allow your code to run. Optionally you can enter the acccount / password under which the code will execute. This will serve as the default for all binary objects is the system.

Bottom of page 290

Once you load a binary file into BO XI, the user interface does not allow for applying an updated version. Rather than delete the entry and recreate the entry again, it's much easier to find the location of the EXE in the FileStore and copy over the latest version. This approach works just as

well for auxiliary files even though the CMC allows you to remove these files and add them again. You can determine the FileStore location by checking the Properties tab in the CMC.

Middle of page 291

The user id and password that you enter here will be used by any schedules you create for this EXE. When a schedule is created it absorbs the user credentials in place at the time and does not check for updated credentials when the scheduled instance is run. Therefore, changing the password in the CMC will have no effect on existing schedules. It is vital, then, to obtain a non-expiring password (and a user id independent of any one individual as they may leave the company) to run the scheduled EXEs otherwise you will need to delete and recreate every schedule in the entire BO XI installation when your password changes.

Bottom of page 295

It would be ideal if criteria screens could be created by extracting the List of Values (LOV, pronounced “love”) variables from the report. Then, even the data needed to populate the filter controls would be accessible through the List of Values objects. Unfortunately this information is not available programmatically through the SDK but such functionality has been requested for the next version.

Bottom of page 362

You can tell BO XI to pull the latest members of an NT group by creating a registry entry and setting the timer value. This timer value will instruct BO XI how frequently to update the membership list of the groups assigned to the objects. You’ll need to create the registry entry yourself as it isn’t created by the install of the product. The registry entry is:

HKLM/SOFTWARE/Business Objects/Squite 11.5/Enterprise/Auth Plugins/<AuthenticationType>/GraphTimeOut

Substitute either secLDAP, secEnterprise, secWinAD, or secWindowsNT for <AuthenticationType>.

The GraphTimeOut key is a string value and the timer value it contains is expressed in minutes. If the value is less than or equal to zero, the default of 15 minutes is used.

Bottom of page 393

Working with WebIntelligence Reports

Most of the reporting examples used in this book cover Crystal Reports. The reason is that BusinessObjects XI was originally Crystal Enterprise, developed by Crystal Decisions for its own reporting technology before it was purchased by BusinessObjects. Therefore the original SDKs do not support WebIntelligence reports. In fact, BusinessObjects XI is the first version of the server product to support the BusinessObjects reporting technologies at all. The Web Services SDK was created by BusinessObjects and is the only SDK to support the traditional BusinessObjects reports.

Though the code used by the SDK can be verbose, it is nevertheless consistent. The code to open a WebIntelligence instance is very similar to the code used to open a Crystal Reports instance. Examine the code in Listing 11-17. This code creates a reference to a WebIntelligence report and schedules it. Setting up the schedule is the same as shown in Listing 11-11.

Listing 11-17 Scheduling a WebIntelligence report

```
Webi oWebi;

oWebi = new Webi();
oWebi = ((Webi) oInfoObject);
oWebi.Name = szReportName;
oWebi.SchedulingInfo = oSchedulingInfo;

oBIPlatform.Schedule(oInfoObjects);
```

Displaying a WebIntelligence report can be accomplished by the code in Listing 11-18.

Listing 11-18 Displaying a WebIntelligence report

```
BIPlatform oBIPlatform;
BusinessObjects.DSWS.Connection oConnection;
BusinessObjects.DSWS.ConnectionState oConnectionState;
ReportEngine oReportEngine;
RetrieveData oRetrieveData;
RetrieveBinaryView oRetrieveBinaryView;
ViewSupport oViewSupport;
DocumentInformation oDocumentInformation;
BinaryView oBinaryView;
Action[] oActions;

oBIPlatform = GetBIPlatform(szServicesURL, szAuthType,
        szDomain, szUser, szPassword);

//Since we'll need the Connection and ConnectionState properties to
//create the ReportEngine class, let's extarct them from BIPlatform
oConnection = oBIPlatform.Connection;
oConnectionState = oBIPlatform.ConnectionState;

//ReportEngine provides a gateway to the report document
oConnection.URL = szServicesURL + "reportengine";
oReportEngine = new ReportEngine(oConnection, oConnectionState);

//The ViewSupport class maps the viewing preferences
oViewSupport = new ViewSupport();
oViewSupport.OutputFormat = sOutputFormatType;
oViewSupport.ViewType = ViewType.BINARY;
oViewSupport.ViewMode = ViewModeType.DOCUMENT;

//RetrieveBinaryView indicates that a binary document,
//as opposed to a character or XML document, should be retrieved
oRetrieveBinaryView = new RetrieveBinaryView();
```

```
oRetrieveBinaryView.ViewSupport = oViewSupport;

//RetrieveData indicate what report data will be returned
oRetrieveData = new RetrieveData();
oRetrieveData.RetrieveView = oRetrieveBinaryView;

//Indicate that when the report is extracted it should be refreshed
oActions = new Action[1];
oActions[0] = new Refresh();

//Extract the report
oDocumentInformation = oReportEngine.
        GetDocumentInformation(szReportCUID, null, oActions, null, oRetrieveData);

//Retrieve binary view of report
oBinaryView = ((BinaryView) oDocumentInformation.View);

//Output report to browser
Response.Clear();
Response.AddHeader("Content-Length", oBinaryView.ContentLength.ToString());
Response.AddHeader("Content-Disposition",
        String.Format("inline; filename={0};", szReportName +
        GetExtension(sOutputFormatType)));
Response.ContentType = oBinaryView.MimeType;
Response.BinaryWrite(oBinaryView.Content);
Response.Flush();
Response.End();
```

A more involved example can be found when setting parameters to a Webi report and displaying it to the browser. Listing 11-19 shows how this is accomplished.

Listing 11-19 Setting Parameters to a WebIntelligence report

```
BIPlatform oBIPlatform;
BusinessObjects.DSWS.Connection oConnection;
BusinessObjects.DSWS.ConnectionState oConnectionState;
ReportEngine oReportEngine;
ResponseHolder oResponseHolder;
GetOptions oGetOptions;
InfoObjects oInfoObjects;
InfoObject oInfoObject;
ViewSupport oViewSupport;
RetrieveBinaryView oRetrieveBinaryView;
RetrieveData oRetrieveData;
RetrieveMustFillInfo oRetrieveMustFillInfo;
DocumentInformation oDocumentInformation;
FillPrompt[] aFillPrompt;
FillPrompts oFillPrompts;
PromptInfo[] aPromptInfo;
DiscretePromptValue[] aDiscretePromptValue;
Action[] aActions;
BusinessObjects.DSWS.ReportEngine.PromptValue[] aParamValues;
BinaryView oBinaryView;

oBIPlatform = GetBIPlatform(szServicesURL, szAuthType, szDomain, szUser, szPassword);

oGetOptions = new GetOptions();
oGetOptions.IncludeSecurity = false;

oResponseHolder = new ResponseHolder();
oResponseHolder = oBIPlatform.Get("cuid://<" + szReportCUID + ">@SI_PROCESSINFO", oGetOptions);
```

```
oInfoObjects = oResponseHolder.InfoObjects;
oInfoObject = oInfoObjects.InfoObject[0];

oConnection = oBIPlatform.Connection;
oConnectionState = oBIPlatform.ConnectionState;

//ReportEngine provides a gateway to the report document
oConnection.URL = szServicesURL + "reportengine";
oReportEngine = new ReportEngine(oConnection, oConnectionState);

//The ViewSupport class maps the viewing preferences
oViewSupport = new ViewSupport();
oViewSupport.OutputFormat = OutputFormatType.PDF;
oViewSupport.ViewType = ViewType.BINARY;
oViewSupport.ViewMode = ViewModeType.DOCUMENT;

//RetrieveBinaryView indicates that a binary document,
//as opposed to a character or XML document, should be retrieved
oRetrieveBinaryView = new RetrieveBinaryView();
oRetrieveBinaryView.ViewSupport = oViewSupport;

//RetrieveData indicates what report data will be returned
oRetrieveData = new RetrieveData();
oRetrieveData.RetrieveView = oRetrieveBinaryView;

//RetrieveMustFillInfo indicates which set of data will be returned on the next trip to the server.
//Here we're looking for the prompts and we'll get this information through a PromptInfo class
oRetrieveMustFillInfo = new RetrieveMustFillInfo();
oRetrieveMustFillInfo.RetrievePromptsInfo = new RetrievePromptsInfo();

//There is only one parameter for this report. Create an array of DiscretePromptValue
//objects and set the value. Here, the parameter corresponds to a country code.
aDiscretePromptValue = new DiscretePromptValue[1];
aDiscretePromptValue[0] = new DiscretePromptValue();
aDiscretePromptValue[0].Value = "US";

//Create a ParamValues array and set the aDiscretePromptValue reference
//to the ParamValue array element
aParamValues = new BusinessObjects.DSWS.ReportEngine.PromptValue[1];
aParamValues[0] = aDiscretePromptValue[0];

//Create an object that will fill the promts for the report
oFillPrompts = new FillPrompts();

//We only need to element as there's only one parameter.
aFillPrompt = new FillPrompt[1];
aFillPrompt[0] = new FillPrompt();
aFillPrompt[0].ID = "ROOT.0";
aFillPrompt[0].Values = aDiscretePromptValue;

oFillPrompts.FillPromptList = aFillPrompt;

//Using the Actions object, assign the FillPrompts object containing the parameter value and
//apply a Refresh method to refresh the data when the report opens to apply the parameter
aActions = new Action[2];
aActions[0] = oFillPrompts;
aActions[1] = new Refresh();

//Retrieve the DocumentInformation
oDocumentInformation = oReportEngine.GetDocumentInformation(szReportCUID, oRetrieveMustFillInfo,
aActions, null, oRetrieveData);
```

```
// Create a reference to the prompts
aPromptInfo = oDocumentInformation.PromptInfo;

aActions = new Action[1];
oFillPrompts = new FillPrompts();
oFillPrompts.FillPromptList = aFillPrompt;
aActions[0] = oFillPrompts;

oDocumentInformation = oReportEngine.GetDocumentInformation(szReportCUID, null, aActions, null,
oRetrieveData);

//Retrieve binary view of report
oBinaryView = ((BinaryView) oDocumentInformation.View);

//Output report to browser as a PDF
Response.Clear();
Response.AddHeader("Content-Length", oBinaryView.ContentLength.ToString());
Response.AddHeader("Content-Disposition",
        String.Format("inline; filename={0};", szReportName +
        GetExtension(OutputFormatType.PDF)));
Response.ContentType = oBinaryView.MimeType;
Response.BinaryWrite(oBinaryView.Content);
Response.Flush();
Response.End();
```