# BIRT Report Object Model – Overview

## Abstract

*BIRT is based on a comprehensive report object model that describes all aspects of the report. The XML design format directly represents the object model. This spec describes the report object model from the developer's perspective.*

## Document Revisions

| Version | Date | Description of Changes |
|---------|------|------------------------|
| Draft 1 | 11/29/2004 | First BIRT release |

## Contents

## 1.   Introduction

The Eclipse BIRT project provides an open, Java-based reporting system with a GUI based on Eclipse. The Report Object Model (ROM) is the report design format that the designer creates and that the Engine executes.

### 1.1   About this Document

This specification describes the all the features planned for release 1 of BIRT, along with many features planned for subsequent releases. See the BIRT Requirements for a high-level statement of what is in the first release. See the details of each element for information on specific elements and properties in the first release.

### 1.2   The ROM Specification Suite

This document is one of several that describe the Report Object Model. This document describes common elements and data types. Other documents, listed below, describe specific parts of the model.

| Document | Content | Status |
|---|---|---|
| ROM Design SPEC.doc | The overall report design. Defines the components of a BIRT design, including parameters. | Firm draft. |
| ROM Base Elements SPEC.doc | The Report Element and Report Item from which all other elements derive. | Firm draft. |
| ROM Layout.doc | Layout rules and elements including free-form and grid | Firm draft. |
| ROM List and Table SPEC.doc | List and table elements | Firm draft. |
| ROM Styles SPEC.doc | Styles | Firm draft. |
| ROM Text SPEC.doc | Text element | Firm draft. |
| ROM Data SPEC.doc | Data sources and data sets | Draft. |
| ROM Scripting SPEC.doc | Scripting | Draft. |
| ROM Page Setup SPEC.doc | Master pages and page sequences. | Firm draft. |

### 1.3   Document Conventions

This specification describes report elements and scripting objects. The document employs a number of formatting conventions to organize the information. Detailed information about elements and objects appears using a variation of the Unix "manpage" structure. The following sections apply to both elements and objects:

**Description**

Provides a description of the element or object.

**See Also**

References related material.

### 1.3.1   Report Elements

The description of report elements is structured by broad element category. Each element is described using a "man page" style structure that includes some or all of the following sections:

**ROM Summary**

Describes the base element and other model-level information.

Name: The informal "display name" for the element. This name uses plain English such as "Report Item" instead of a computer literal such as "report-item".

Base element: many element definitions derived from another element definition. This means that the derived element inherits all the properties of the base element. Note that definition inheritance is different from element inheritance. Definition inheritance is shorthand for defining common properties; element inheritance is shorthand for setting properties. Definition inheritance is part of the model definition; element inheritance is part of a specific report design.

Availability: Describes when the element was added to BIRT. Since BIRT has not yet been released, this item identifies if the item is planned for the first release, or a later release.

**XML Summary**

Describes how the element is represented in XML. Elements are either *abstract* or *concrete*. Abstract elements are represented using an XML schema *complex type*. Concrete elements are represented using an XML element (tag name).

Element: Gives the element name for a concrete element.

Complex type: Many elements are defined using an XML schema complex type.

**Scripting Summary**

Report elements are made available at runtime through JavaScript objects. Generally, an element has two associated objects: one that provides access to the design information about the object, and another that provides information about the specific *instance* that holds run-time state.

Design Object: Gives the name of the design-time object. In most cases, an object defined in an abstract element is used for all derived elements.

Runtime Object: Gives the name of the object that holds runtime state for the element. Again, often one JavaScript object holds the state for a number of different elements.

**Properties**

Summarizes the properties for the element. Each property is then described again in detail in a subsequent section.

**Methods**

Summarized the methods for the element. A method is simply a place to put JavaScript code. The method is then described again in detail in a subsequent section.

**Contents**

Many elements contain other elements. This section summarizes each content element. The elements are described again in detail in subsequent sections.

### 1.3.2  Report Element Properties

Report elements define a set of properties. Properties have three different manifestations: as part of the XML design file, as part of the abstract Report Object Model, and as part of a scripting object.

**ROM Summary**

ROM defines elements and properties using a "user-friendly" descriptive name and type. This section gives that information:

Name: The ROM name for the property. This name is usually capitalized and can contain spaces.

Type: The ROM type for this property. There may be many ROM types that map to a single XML type. For example, ROM differentiates between an element name, an element reference and a column name. However, all of these are simply strings in XML.

Expression type: If the property itself is an expression, this entry defines the expected return type of the expression.

Inherited: Most ROM properties are inherited by derived elements, but some are not. This entry appears only for elements that derive from the `ReportElement` base element.

Default value: Explains the default value that appears if the property is not set and is not inherited.

Required: Appears for those few properties that must be set. Most ROM properties provide a default value.

Availability: Describes when the element was added to BIRT. Since BIRT has not yet been released, this item identifies if the item is planned for the first release, or a later release.

**XML Summary**

Describes how the property is implemented in the XML design file. Many properties appear as attributes on the element, though some appear as separate elements. In general, a property is implemented as an XML attribute unless it has complex internal structure, or is contains characters that cannot appear in an attribute.

Attribute name: Gives the name of the attribute if the property is implemented as an attribute.

Element name: Gives the name of the element name if the property is implemented as an element.

Type: Gives the name of the XML type for the property. This can be a standard XML schema type (string, integer, etc.), a simple XML type (for attributes) or a complex XML type (for elements).

**Scripting Summary**

Properties are available at runtime using the two JavaScript objects mentioned above. This section explains how the properties appear in JavaScript.

Property name: The name of the JavaScript property within the design and runtime objects. Reference the property as `obj.propName` in code, where "`obj`" is a variable that points to the object, and "`propName`" is the name of the property shown here.

Type: The type of the property in JavaScript. JavaScript is dynamically typed, so this type explains the kind of value returned by the property. Many properties have separate runtime and design time types.

Settable at runtime: Some properties can be set via code at runtime. The new value applies to just that one runtime instance. Other properties are design-only and cannot be set at runtime. When a property can be set, the script does so using the property name on the runtime object.

### 1.3.3  Element Method

An element method is one that the user implements to provide custom report behavior.

**Synopsis**

Provides a description of the method, including any arguments it may take.

Availability: As described above.

**Arguments**

Defines any arguments passed to the method.

**Returns**

Some methods return a value. This section describes the return value.

### 1.3.4  Slot

Some elements act as containers for other elements. Some elements, such as a report design, can contain different kinds of elements. Each group of contained elements is called a slot.

**ROM Summary**

Name: As above.

Cardinality: Whether one or multiple elements can appear within the slot.

Required: If required, then at least one element must appear within the slot.

Availability: As above.

**Contents**

Summarizes the elements that can appear within the slot.

**XML Summary**

Element name: the name of the element that encloses the contained elements.

**Scripting Summary**

Property Name: the name of the property that provides the element definitions

Design type: The object type for the element designs.

### 1.3.5  JavaScript Objects

**Synopsis**

Provides a summary of the item

**Constructor**

Describes whether the application can create an instance of this object. Many objects are created by BIRT itself, others can be created by the application. If the application can create an instance of the object, this section describes the constructor method.

**Properties**

Lists the properties defined for the object. If the object is one that describes a report element, then a generic entry may appear that directs the reader to the scripting summary for each element.

**Methods**

Summarizes the methods available for the object.

**Static Methods**

Summarizes static methods available for the object. These are functions that are part of the object definition, but don't work with any specific object. See a JavaScript reference for more information on static methods.

**Example**

Provides a code example for using the object.

### 1.3.6   JavaScript Object Method

The method section provides detailed information for how to use the method.

**Synopsis**

Summarizes the method arguments and return value (if any.)

**Arguments**

Describes the method arguments in detail.

**Returns**

Describes the method return value in detail.

**Example**

Provides a code example for using the method.

- Base element: identifies the base element (if any) and states how the element uses properties inherited from its base element.

- Properties: Elements provide properties that let the user customize the element. Properties are name/value pair, with the value being one of the ROM types defined earlier.

- Style Properties: All style properties are defined on the Style element, but are used by the various visual report items. This section explains which style properties apply to each visual item, and how they apply.

- Contents: Many elements contain complex structure including other elements. Content can be an element, a list of elements, or a list of some simple structure. Each of these elements or structures are described with their own element section.

- Scripts: Many elements provide scripts that the user can define to customize the element. If the script is present, the Factory or Presentation engine calls the script at the point in time defined by the script

## 2.  Technical Overview

This section presents a general overview of the Report Object Model. Later sections explain specific report elements and scripting objects.

### 2.1  XML Design Format

ROM is implemented as an XML schema. The open, text-based structure of XML allows ROM to achieve a number of important goals. First, third parties can easily extend ROM with new report elements. Second, developers often work with source control systems such as CVS. Often, developers need to identify changes to the application between versions. The text-based XML format makes this easy; allowing BIRT to integrate well into existing development workflows.

### 2.2  Overview

BIRT report designs contain two broad categories of information: visual and non-visual. Non-visual information includes scripts, data sources, data sets (queries), included libraries and so on. Visual information divides into two categories: master pages and content. Master pages describe the "page setup" for the report, while content describes the material that is to be "flowed" onto a series of pages. Visual information further divides into two additional categories: style and structure. Style describes colors, fonts, alignment, borders and other purely visual aspects of the design. Content describes the structure of the data: as a table, a list, a chart, etc.

A BIRT design is structured as hierarchy of *report elements*. The term *element* simply means an object that appears as part of the report. BIRT provides a wide range of report elements including data sources, styles, visual elements and more. Elements exist to represent each of the kinds of visual and non-visual information described above.

The BIRT model allows the report developer to build a wide variety of reports from the simplest tabular list to the most sophisticated document that may include multiple lists, tables, charts, text blocks and other content. For example, a sales summary report may have show a list of closed sales, a list of pending sales and a list of lost sales. Common parameters can tie together a sequence of sections. For example, in the sales summary report, limit the lists to those deals in a particular region.

#### 2.2.1  Data Access

Virtually every report presents data obtained from outside the report itself. Typical data sources include relational databases, XML files, Java classes and so on. Many data sources require some kind of *connection*, such as a connection to a relational database. The BIRT *data source* element represents connections. BIRT reports work with tabular *result sets* produced by *data set* elements. Typical data sets include SQL queries, stored procedures, and so on. The developer can also create custom data sets using scripting.

See the ROM Data Specification for details on data sources and data sets.

#### 2.2.2  Visual Structure

The visual structure of a BIRT report is given by one or more top-level *sections*. A section can include a traditional grouped *list*, a grouped *table*, a *matrix* or a *chart*. Further, the developer can create advanced designs using *containers* such as *grid*, *text* or *free-form* sections. Sections are simply one kind of report item.

Many sections contain *report items* to present data, images, graphics and more. A report item is simply a specialized form of a report element that add visual layout attributes such as size, position, style and so on. Lists, tables, matrices, and grids define structures with specific places to insert report items. Lists and tables are divided into horizontal *bands*, and each band can contain another section, called a *subsection*.

Once the developer creates the sections and report elements, he may want to select which elements should appear for a given set of conditions. For example, the sales summary example above may provide parameters for choosing which lists to include, depending on what the user wants to see.

The visual part of ROM is fully described in the following specifications: Layout, List and Table, Text, and Simple Elements.

### 2.2.3  Reuse

Typical applications consist of not just one report, but a suite of related reports. The reports provide different views of the data, tailored to different audiences and timeframes. The reports tend to have a large degree of overlap: they use the same visual styles, work with the same data sources, use the same business logic and so on. The ability to reuse material across reports becomes vital to ensure consistency, speed development and reduce maintenance costs. While copy & paste reuse may help with development, it does little to reduce maintenance costs.

Therefore, BIRT includes a number of features to help report developers work efficiently with a suite of related reports.

- *Libraries* allow developers to define common styles, data access, business logic, images, text and more. Reports include libraries and reuse the common elements.

- Java integration allows report developers to use business logic that already exists within an application. For example, many applications provide a data access layer that wraps the underlying database. BIRT reports can use such objects as data sources.

- *Styles* allow the report developer to define a common visual look for a suite of reports separate from the actual content of the report. The BIRT style system is very similar to the well-known CSS style system.

### 2.2.4  Scripting

Reports are much more than a query and a visual layout. Data is seldom in the exact form needed by the report. The people who request reports often have complex requirements for how the data should appear within the report. No matter how many options and properties ROM may provide, report developers will still find reports that can't be done using just the ROM itself.

*Scripting* allows developers to add custom code to a report design. Scripting lets the developer work with the most poorly-structured or complex of real-world data sources. Scripting lets the developer insert business logic within the report; either to manipulate the report data, or to adjust report content conditionally. Scripting also allows the developer to call into existing business logic written in Java.

BIRT scripting is based on ECMAScript popularly known as JavaScript in browsers. BIRT uses the Mozilla Rhino JavaScript engine. Rhino provides excellent integration

with Java classes, allowing BIRT scripts to seamlessly call application logic written in Java.

## 2.3 Leverage Open Standards

The Report Object Model (ROM) is designed to leverage the Cascading Style Sheets (CSS) and XSL Formatting Objects (FO) standards. In general, the style information about a design is passed directly from the design to the target format: HTML with CSS or FO. The *User Agent* (UA) handles detailed visual formatting. On the web, the user agent is the browser. With FO, the user agent is the FO processor that converts FO into PDF, RTF, Excel, print or some other format.

Said another way, ROM is a description of how to use database information to produce an HTML & CSS document on the one hand, and FO on the other hand. Because these are open standards, we assume that users will be more familiar with them than with a BIRT-defined alternative. If a developer needs to do some very detailed formatting, then he should be able to leverage his knowledge of CSS.

Said yet another way, the ROM does not view HTML & CSS as a tool to render BIRT's own visual model. Rather, ROM defines a visual model that is designed to render smoothly into HTML and CSS. ROM's concepts and terminology are carefully crafted to build on HTML and CSS.

Because FO was also designed to be compatible with CSS, FO provides an excellent target format for print. ROM visual concepts and properties are designed to map smoothly into FO.

## 2.4 Non-Document Formats

Although open standards are important, BIRT is designed to render to many different output formats including Excel, hand-helds and more. Some elements may work only in the Excel environment, others only in the hand-held environment. Although we expect print and web to be an important target, BIRT is not at all limited by these environments.

## 2.5 Document Focus

The earliest reporting systems in the 60's, 70's and 80's were character based: people created reports by laying out fields on a fixed-width character grid. Tab stops were used to align content.

In the early 90's, graphic technology became mainstream and reporting tools adopted a new model: that of a graphic device. The early graphics environments focused on pixel-based layouts. Applications displayed contents by painting text, images and graphics at pixel offsets on the screen or page. Reporting tools designed at that time and have a strong "drawing" style layout model as a result. Users build reports by dragging and dropping components much as one would build a drawing.

The drawing-based model allowed much more flexibility than the older character-based model. However, experience suggested reports are primarily documents, and the best reports are designed using document layout rules. For example, while a user can put fields anywhere, reports look best when fields are laid out in rows and columns, or when the layout consists of blocks of lines. Data is often presented in tables of rows and columns.

At the same time, the web has gained importance as the preferred way to find and present information. The web is based on HTML, which is fundamentally a document

standard. A web author uses HTML to indicate the structure of the document along with style information. The browser then creates a graphic representation of the document based on the characteristics of a given device, user display preferences (such as font size) and so on.

ROM builds on this insight by providing layout "containers" that help the user create a well-designed report. Grids and tables help organize fields. The Text item does normal text-style layout, even when the text contains formatting. The developer specifies the type of layout that is wanted, and BIRT works out the details of how that layout is best achieved on any given device or output format.

Of course, ROM still provides pixel-based positioning for those reports that need it. Such formatting does, however, require the developer to deal with the differences in font metrics and display resolution across devices.

Experience suggests that for every one report that must have pixel-accuracy in reproducing, say, a government form, there will be dozens that simply need to look great on the web. Hence, document-style layout is the primary focus of ROM, while drawing-style layout is supported for those cases when it is required.

## 2.6  Extensibility

ROM defines a rich set of reporting capabilities that are widely applicable to many types of reports. Specific applications may need additional capabilities unique to that application domain. ROM provides many extension points to support these specialized needs.

- A custom visual element framework allow developers to add their own visual elements into the reporting model. For example, a particular application may need to work with a custom charting or data visualization system.

- Extensible data adapters allow developers to add data sources that connect to business applications, specialized data sources, SOAP services, Java classes and more.

## 3.   ROM Overview

This section provides an overview of the BIRT Report Object Model. See the detailed specifications for details of each component.

ROM divides report elements into two broad categories: visual report items, and non-visual elements. Visual items include things like lists, tables and labels. Non-visual items include data sources, data sets, styles and more.

## 3.1  Approaches to Report Design

There are two popular ways to build a report. People familiar with data and SQL may prefer to start by defining a query, test the query then move to layout. People more familiar with layout may prefer to first define the report layout, then later go back and figure out the data needed for that layout. BIRT supports both. However, in the discussion that follows, we will focus on the data-driven approach.

## 3.2  Data Sources and Data Sets

A report can start with a *data set*, an element that specifies how to obtain tabular data to display within the report. Most data sets require a *data source* which is a connection to an external *data provider* such as a database.

ROM data sets support input and output parameters. Input parameters represent a parameter in an SQL statement, and input parameter for a stored procedure and so on. Stored procedures can provide output parameters. Some data sets, such as stored procedures, support multiple *result sets*.

Information about the data set is called *metadata.* The design file can include a complete description of the metadata. Many data providers can provide the metadata, and in this case it need not be duplicated in the design file.

See the *ROM Data Specification* for details.

## 3.3  Layout

The next step in building a report is to provide the report *layout*. A report layout divides into one or more *sections*. A section is simply a portion of a report similar to a paragraph or chapter in a printed document. A report is a series of sections that follow one another down the page or across pages.

ROM supports many different layout styles within a section. Most reports use a series of standardized layout rules. ROM simplifies report design by providing report items that assist with typical layouts. Data to appear in tabular format uses a ROM Table item. Data to appear as paragraphs of text use the Text or Multi-line Data item. Sometimes a report wishes to display data in a "label: value" format in which the labels and values are aligned in columns. The Grid element assists with such a layout.

Some reports need very fine control over placement of items, perhaps to mimic a predefined printed form. Such reports use the free-form element which allows the report developer to position items anywhere within a section.

Some reports provide a sophisticated layout then repeat that layout for each row in a query. For example, a company may print a report that contains a long series of invoices, one for each active customer. The ROM List element provides the ability to iterate over a data set and produce sections for the first row, each row or last row.

Sections organize report content. The content itself consists of text, graphics, charts, and so on. Each item provides one or more expressions to *bind* the item to data form the data set.

See the *ROM Layout Specification*, *ROM List and Table Specification*, *ROM Simple Items Specification* and *ROM Text Specification* for details.

## 3.4  Styles

Reports are documents, and document design is frequently based on a set of styles applied consistently throughout the report and across a set of reports. BIRT provides a style system closely related to the Cascading Style Sheet standard. The developer can establish a set of styles then apply the styles to the report layout. If style needs change, reports can be updated simply by changing the *style sheet*. See the *ROM Styles Specification* for details.

## 3.5  Scripting

While expressions and standard report elements are sufficient for many reports, some require complex business logic or specialized layout rules. Scripting allows the report developer to add any amount of extra, application-specific logic needed to accomplish a specific reporting task.

Scripting is often needed to compute data values using specific business rules. BIRT recommends that the developer implement this logic as *computed columns* in the data set. Doing so allows the developer to test the logic before building a report layout around the logic.

See the *ROM Scripting Specification* for details.

## 3.6  Report Parameters

Reports often offer the user a range of customization options. For example, a customer information report may let the user select the specific customer, the amount of detail to include, a time range, and so on. *Report parameters* are the way that the user provides such selections to the report at runtime. Report parameters define a user interface for the end-user. Parameter values then appear as values that the report can use in expressions, as data set parameters and in custom scripts. See the *ROM Parameters Specification* for details.

## 3.7  Master Pages

Report layout controls how data is presented in a report. Report developers also need to control the look of the overall page; including margins, header & footer, background images and so on. The *master page* is the mechanism for page set. See the *ROM Master Page Specification* for details.

## 3.8  Reuse

Applications typically include a related set of reports that present different views of data; perhaps at different levels of detail, over differing time periods, for different audiences. However, a suite of reports generally has large amount of commonality: the same data sources, styles, business logic and so on. BIRT provides *libraries* to define reusable report content. (Libraries are not in the first release.) See the *ROM Library Specification* for details.

## 3.9  Localization

Today's business is global: customers, employees and suppliers reside in different countries. Each may prefer to see reports in their own language and locale. BIRT supports two levels of localization. First, BIRT itself can be localized. Second, a report can be designed to be self-localizing depending on the language and locale of the person using the report. The report designer can externalize all user-visible strings, then have the strings localized. BIRT looks up the proper localized string at runtime. See a section later in this specification for details.

## 3.10  Configuration Variables

Reports need many kinds of information to run. There is the information in the report design itself provided by the report developer. There is run-specific information provided by the end-user though report parameters. Finally, there is configuration information

provided by the system administrator. For example, a report developer may use a test database during development, but the deployed report will use a production database.

*Configuration variables* provide a way to externalize configuration data so that the data can vary depending on the environment. For the database configuration above, a configuration variable may provide the database server name. See the *ROM Customization Specification* for details.

## 3.11  Extensibility

BIRT is designed to integrate well with an overall Java environment. Applications have specialized needs that are sometimes best met with specialized extensions to the reporting model. ROM addresses this with a number of extensibility solutions.

- Applications can add custom business logic though scripting or customized Java classes.

- Applications can customize existing components by adding properties and custom logic.

- Third parties can add new components to BIRT though an extensible component framework. (Specification not yet released.)

- Third parties can add new kinds of data sources though a data access extension framework. (Not part of the first release.)

See the *ROM Customization Specification* and the *ROM Scripting Specification* for details.

## 4.  Working with ROM

Most users will use the Eclipse Report Designer (ERD) to build reports using ROM. Java developers can use the BIRT Design Engine (DE) interfaces to build a report though code. Scripts use a rich set of JavaScript objects to interact with the report design and report instance at runtime. Report developers can use any text-based "diff" facility to compare two report designs, or two versions of the same design.

The ERD is described elsewhere; this specification focuses on the conceptual description of ROM and its implementation in the XML file and in report scripting.

### 4.1  Overview of the ROM XML Schema

The ROM schema is based on a three concepts:

- A design element identified with an XML element, and with a series of properties and/or methods.

- A property represented using a "property" element in XML.

- A structure that holds the value of properties that consist of lists of items.

- A slot that holds other report items.

For example:

```
<free-form name="sample">                                    Element
   <report-items>                                            Slot
     <label name="myLabel">                                  Element
```

```
        <property name="color">red</property>              Property
        <property-list name="visibility">                  List property
          <structure>                                      Structure
            <property name="format">html</propery>         Structure member
            <property name="expr">true</property>          Structure member
          </structure>
        </visibility>
      </label>
    </report-items>
  </free-form>
```

This specification describes the design elements, properties and structures that make up ROM and hence the XML schema.

## 4.2  Overview of Scripting

Developers can provide methods and expressions associated with elements. Each of these is a script written in JavaScript.

### 4.2.1  Expressions

An expression is a property that can accept a JavaScript as its value. The expression can contain any amount of JavaScript code, but must return a value:

```
var temp1 = row.col1 * row.col2
var temp2 = row.col3 / row.col4
temp1 – temp2
```

In JavaScript, the value of an expression is the last value computed in a script. See the *ROM Scripting Specification* for details.

### 4.2.2  Methods

A method is an optional, named script that the BIRT Engine will call at defined points while the report is run or viewed. Methods allow the programmer to customize the report. For example, a data set can override the "beforeOpen" method to perform custom setup before opening a data set.

Most methods do not return values. Methods are implemented as methods on a JavaScript object. Methods have access to the "this" variable that points to the runtime representation of the element that defined the method.

## 4.3  Extensions

Each extension, such as charting or extended data sources, is implemented using its own XML syntax, scripting objects, model API and so on. While extensions are free to adopt the same XML syntax and scripting conventions as ROM, they are not required to do so. Extensions are also not required to use the ROM-defined ways of working with data sets, using styles, using inheritance and so on. This allows extension writers to use their own creativity to adopt solutions that best leverage their own experience and customer needs.

# 5.  Data Types

ROM elements are defined by *properties*. Each property has a data type. The data type can be simple (string, integer) or it can internal structure. A property type can be a *structure*, meaning that the property is composed of a collection of additional properties.

For example, an Action (hyperlink) property type is composed of a target window, a URL, and so on. A property type can also be a *list*, such as a list of custom colors, a list of visibility rules, and so on.

ROM combines these types to produce a rich report description system. A property can contain a list of simple values, or a list of structures. Structures can contain additional lists or structures. ROM applies these techniques to create a definition suitable for each kind of property. It also allows ROM to treat all properties uniformly, whether they be simple, structures, or lists.

## 5.1  Simple Data Types

The following lists the data types used within this specification. The JavaScript types below are the types of the design-time information. See the specification if there is an additional run-time type. (For example, an expression is a string at design time, but will evaluate to a string or integer or other value at runtime.)

| ROM Type | JavaScript Type | Description |
| --- | --- | --- |
| Boolean | Boolean | A simple true/false value. |
| Choice | String | The value must be selected from a predefined list of choices. |
| Color | String | An RGB color value or color name. |
| Column | String | Reference to a column in a data set. |
| Date/Time | Date | A date/time value. |
| Dimension | String | A CSS dimension. The units are assumed to those defined in the Units property of the report. The units are expressed as a floating point number such as "1.25". To override that default, specify a suffix as one of "in" (inches), "mm" (millimeters), "cm" (centimeters), "pt" (points), "pc" (picas), "%" (percent), "em" ("em" size for the current font) or "ex" ("ex" size for the current font.) |
| Element Reference | String | A reference to another element in the design. |
| Element Name | String | A name that follows the BIRT naming rules. Used to identify the name of report elements. |
| Expression | String | An BIRT expression using JavaScript and the BIRT extensions. |
| Extends | String | A reference to a parent element of the same type as the derived element. |
| Float | Number | A decimal number such as "1", "1.33" or "2.5." |
| Integer | Number | An integer in the range 0 to $2^{32}$ - 1. |
| Name | String | Identifies a property within a structure that provides the name to be used for associative lookup in property lists. (See below.) |

| ROM Type | JavaScript Type | Description |
|---|---|---|
| Number | Number | An arbitrary-precision decimal number. Is accurate enough to represent the largest currency amounts down to two decimal places. |
| Points | Number | A measurement expressed in points. |
| String | String | A string value. The string cannot be externalized. |
| String List | Array | A list of non-externalizable strings. |
| Structure List | Array | A property made up of a list of *structures*. A structure is a set of named properties. |
| URI | String | A URI of the form "http:", "file:", "iServer:", or "mailto:". |
| Variant | Object | A value that can hold any of the above. |
| XML | String | Text in XML format. Is often a fragment of XML rather than a complete file. |

### 5.1.1  Scripting

Properties with simple data types are made available in scripting using JavaScript properties. All properties are available on a design definition object for an element. The design definition element simply provide access to the design which the user created. A script accesses the simple property by first obtaining the definition object (as described elsewhere), and simply requesting the property values. For example, to get the color of an element:

```
var theColor = elementDefn.color
```

Scripts can access, but not modify, the design-time value of a property.

Many properties also have a run-time aspect. As the report runs, the Factory creates *instances* of each element that describe the actual report being created. All properties are available on the JavaScript object that represents the instance. For example:

```
element.color = "red"
var theColor = element.color
```

However, the property may exist in a different form. This is easiest to see for an expression. The design-time information provides the text of the expression, while the run-time object provides the evaluated value of the expression.

Many properties can be set on the runtime object, but not all. The description for each property describes which can be set, and which cannot.

## 5.2  Dimensions

A *dimension* is a physical measurement. Dimensions always have a unit. Supported units include the following defined in CSS:

| Unit Suffix | Meaning |
| --- | --- |
| (None) | Use the design's default units. (See below.) |
| cm | Centimeters |
| in | Inches |
| mm | Millimeters |
| pt | Points (1/72 of an inch) |
| pc | Pica (1 pica is equal to 12 points) |
| px | Pixels. (For print, see the CSS spec. A pixel is about 0.21 mm.) |
| em | ems, the height of the element's font |
| ex | x-height, the height of the letter 'x' |
| % | Percentage. The definition of each property identifies the container property from which the value is computed. |

### 5.2.1  Default Units

Most developers work with just one dimension. For example, US developers usually work in inches; most others work in cm or mm. As a convenience for hand-created designs, the developer can omit the dimension suffix if the units are in the *default units* for the design. The default units are set using the Units attribute of the report design element.

## 5.3  Property Lists

A list property is defined as being a "list of x" where x can be one of the simple data types, or one of the structures defined below. A list property is represented as an array in JavaScript. For example, the list of custom colors for a report design is represented by a list property. The following obtains the structure that defines the third custom color:

```
var colorDefn = designDefn.palette[ 2 ];
```

A property list is composed of a series of values. In general, the values appear in the list in the same order that they appear in the report design XML file, which generally reflects the order in which they were created in the UI.

### 5.3.1  Associative Lists

All property lists allow numeric indexes. Some also allow associative lookup using the name of an item in the list. For example, to look up the custom color: "seaGreen":

```
var colorDefn = designDefn.palette.seaGreen
```

or,

```
var colorDefn = designDefn.palette[ "seaGreen" ]
```

The first form can be used when the name to be looked up is known when the code is written, and follow the JavaScript identifier naming rules. The second form can be used

to look up a value when the value is not known ahead of time, or when the name does not follow JavaScript identifier rules. For example:

```
var colorDefn = designDefn.palette[ params.colorName ];

var colorDefn = designDefn.palette[ "Sea-Green" ]
```

The associative lookup is provided when:

1. The list contents are structures, and

2. Exactly one of the properties within the structure is of type Name.

## 5.4  Property Structures

Some properties are composed of multiple values. For example, an Action (hyperlink) property has a URL, a target window name, and more. A property structure allows a single property to provide a set of related values. A property structure is just like a structure in C or C++, and is similar to an object in Java or JavaScript. Structures have no members, they just have a set of properties.

Structures also have a name. ROM uses the structure name to determine the properties that make up the structure. Some structure names are "Text", "Action" and "Color Definition".

### 5.4.1  Scripting

The JavaScript object `PropertyStructure` represents a property value that is a structure. This object is simply a convenience so that the JavaScript dot-notation can be used to access the properties that make up a structure. For example, assume that an element has a property called "element" that is of the Action type mentioned above. We can then access the parts of the action as follows:

```
var windowName = elementDefn.action.targetWindow;
var theLink = elementDefn.action.url;
```

This specification identifies each of the ROM-defined structures. As described elsewhere, extension builders can add custom structures. Custom structures work identically to ROM-defined structures.

Each *member* the structure is defined exactly like a property of an element. The member can be a simple type, a list, or another structure.

## 6.  Structured Data Types

ROM defines the following structure data types:

| ROM Type | Description |
|---|---|
| Action | Defines a hyperlink. |
| HTML | A string with optional embedded HTML formatting tags. Can be externalized. |
| Parameter Binding | Represents an association between an expression and a data set or nested report input parameter. |
| Search Criteria | Defines a search criteria for a drill-through hyperlink. |
| Text | A string that can be externalized. Such properties have two parts: the default text and the message key. |
| Visibility Rule | Identifies the format in which to hide the element. |

In each case, the JavaScript type of the property is the `PropertyStructure` object discussed above.

## 6.1  Text and HTML Structures

Provides a string value defined within the report along with an optional resource key to localize the string.

**Summary**

Display Name: Text or HTML

Availability: First release

**XML Summary**

For externalized text:

```
<text-property name="propName" key="resourceKey>
   text
</text-property>
```

For externalized HTML:

```
<html-property name="propName" key="resourceKey>
   text
</text-property>
```

Alternative form for a text property with no externalization:

```
<property name="propName">text</property>
```

In each case, *propName* is the name of the property.

**Design Object Scripting**

```
elementDefn.propName.text
```

```
elementDefn.propName.key
```

**Runtime Object Scripting**

```
element.propName.text
```

```
element.propName.key
```

**Properties**

```
text
```

Default text if no resource key is provided, or if the translated text cannot be found.

```
key
```

An optional string resource key that identifies a message in a resource bundle. Allows the message to be translated.

**Description**

Many elements within a report require static text: labels, chart legends, parameter prompt strings and more. BIRT allows the developer to externalize all user-visible text. Every Text and HTML property has two parts: default text and a resource key.

The resource key is a string name that identifies the actual message text in an external message catalog. BIRT uses the current language, along with the resource key, to find the text. In this context, a message is a specialized form of a *resource*: information kept outside the report to ease localization.

The default text appears in the design environment, when the resource key is blank, and when the message cannot be found in the resource bundle.

Customers who create reports for just one language will enter the message text directly as the default text.

The difference between the two properties is that Text is plain text. Any HTML-like characters are treated just like text. However, the HTML type can contain additional formatting in the form of HTML.

An example of a Text property value is:

```
Warning: Your sales are < $1234!
```

An example of an HTML property value is:

```
<b>Warning</b>: Your sales are &lt $1234!
```

**See Also**

The Resources element within the Report Design element.

### 6.1.1 `text` Property

Default text if no resource key is provided, or if the translated text cannot be found.

**Summary**

Display Name: Text

ROM Type: String

JavaScript Type: String

Default value: None

Settable at runtime: Yes

Availability: First release

**Description**

Static text for the property. This text can be plain text, or HTML as defined by the particular property that uses this structure. Enter text for this property if your report will not be used in multiple languages, or if you want a default text in case the user's language does not provide a translation.

### 6.1.2 `key` Property

Resource key for externalizing the property value.

**Summary**

Display Name: Resource Key

ROM Type: String

JavaScript Type: String

Default value: None

Settable at runtime: Yes

Availability: First release

**Description**

This resource key, if provided, names a resource in a *resource bundle*. The resource bundle can appear within the translations property of the report design or report library, or can appear in an external resource bundle file.

**See Also**

`DesignDefn.translations` property

## 6.2 `Action` Structure

Defines a hyperlink.

**Summary**

Display Name: Action

Availability: First release (with limitations as shown below)

**Properties**

`linkType`

The type of link: hyperlink, bookmark link or drill-through.

`targetWindow`

The name of the target browser window for the link. (Optional.) Used only for the Hyperlink and Drill Through options, ignored for the Bookmark Link option.

`hyperlink`

The expression that provides the actual link.

bookmark

    The target bookmark within this report.

reportName

    The name of a report.

parameters

    Optional list of parameter bindings for the target. Used when the hyperlink is used to run a report.

search

    Optional list of search criteria.

**Property Usage**

| Property | Link Type | | |
| --- | :---: | :---: | :---: |
| | **Hyperlink** | **Bookmark Link** | **Drill-through** |
| targetWindow | ✓ | | ✓ |
| hyperlink | ✓ | | |
| bookmark | | ✓ | ✓ |
| reportName | | | ✓ |
| parameters | | | ✓ |
| search | | | ✓ |

**Description**

The Action structure defines a hyperlink. There are five major kinds of actions:

- Hyperlink: To an external web site, e-mail system, etc.

- Bookmark Link: To a location within this report as specified by a bookmark.

- Drill-though: To a location within another report as specified by a bookmark.

The drill-though link can link to an existing report, or request to run a new report. It can optionally include:

- Parameters to pass when creating a new report.

- A location within the target report as specified by a search.

- A location within the target report as specified by a bookmark.

The Hyperlink property returns a standard web-style link with "http:" or "mailto:" prefix. Expression hyperlinks can include special BIRT features (search and so on), but only as encoded into a URL. If the link wants to jump to a specific target within the document, encode that target within the URL as defined by target document type. (For example: http://foo.com/docs/myDoc.html#myTarget.)

If the user provides a relative URL, it is assumed to be relative to the same web server that is being used to view the report. The exact semantics are implementation-specific in

the open source release. In the commercial release, the URL is relative to the web server that hosts ActivePortal.

A bookmark link simply identifies a bookmark identified within this report. Use the Bookmark property of a report item to create the target bookmark.

### 6.2.1  linkType Property

The kind of link: hyperlink, bookmark link or drill-though.

**Summary**

Display Name: Link Type

ROM Type: Choice

JavaScript Type: String

Default value: Hyperlink

Settable at runtime: Yes

Availability: Only the Hyperlink form is supported in the first release.

**Choices**

| Display Name | Internal Name | Description |
| --- | --- | --- |
| Hyperlink | hyperlink | A link to a web page, "mailto:", or other web resource. |
| Bookmark Link | Bookmark-link | A link to another location within this same report. |
| Drill-Though | drill-through | A link to another report. |

**Description**

This property identifies the kind of link. The link type determines which link properties to use as shown above. The default is a hyperlink.

A drill-through is a link to another report. The report name is given as a relative or absolute name. If the name is relative, then the target document is on the same web (application) server as the current report. For example:

- report1: in the same virtual directory as this report.
- ../report2: Up one virtual directory from this report.
- ../summary/report3: In a sibling directory to this report.
- /reports/summary/report4: In an absolute location on this web server.

The destination can be any type of report as supported by the web server interface that receives the link commands. If the destination is a BIRT design, the report will be run.

### 6.2.2  targetWindow Property

The name of the target browser window for the link.

**Summary**

Name: Target Window

ROM Type: String

JavaScript Type: String

Default value: Blank. (The link replaces the report in the current browser window.)

Settable at runtime: Yes

Availability: First release

**Description**

The name of the target browser window for the link. Used only for the Hyperlink and Drill-Through options, ignored for the Bookmark Link option.

### 6.2.3  `hyperlink` **Property**

Simple web hyperlink.

**Summary**

Display Name: Hyperlink

ROM Type: Expression

Expression type: String

Default value: None

Settable at runtime: Yes

Availability: First release

**Description**

The expression that provides the actual link. This is a Web-style hyperlink and includes http:, mailto: and other URLs. Both relative and absolute URLs are allowed. The meaning of relative URL depends on the deployment environment.

If the property is not set, then no link is defined, even if the link type is set to "Hyperlink."

### 6.2.4  `bookmark` **Property**

Target bookmark defined within this same report, or another report for a drill-though link.

**Summary**

Display Name: Bookmark

Type: Expression

Expression type: String

Default value: None

Settable at runtime: Yes

Availability: After the first release

**Description**

The target bookmark within this report.

If the link type is bookmark link then the value here must match a bookmark defined within the report. If the bookmark target is not defined, then the link will do nothing.

If link type is drill-though link then the value here must match a bookmark defined within the target report. If the bookmark target is not defined, then the drill-though link will display the first page of the report.

**See Also**

Report Item `bookmark` Property

### 6.2.5  `reportName` Property

Name of the target report for a drill-through link.

**Summary**

Display Name: Report Name

ROM Type: String

JavaScript Type: xx

Default value: None

Settable at runtime: Yes

Availability: After the first release

**Description**

The name of a document on the same server as this report. Can include relative or absolute names. If the suffix is omitted, it is assumed to be 1) a BIRT document or 2) a BIRT design. If the report name is blank, then the no hyperlink is defined even if the link type is set to Drill-though.

### 6.2.6  `parameters` Property

Optional set of parameter bindings when running a report in a drill-though link.

**Summary**

Display Name: Parameter Bindings

ROM Type: List of Parameter Binding structures

JavaScript Type: Array of `PropertyStructure` objects

Default value: None

Settable at runtime: Yes

Availability: After the first release

**Description**

Optional list of parameter bindings for the target. Used when the hyperlink is used to run a report. The bindings evaluate an expression within the current report, and bind that value to a report parameter in the target report. For example, if the current report is a list of customers, the report can include a hyperlink that runs a report that shows any open orders for that customer. In this case, the parameter bindings pass the customer ID to the target report.

### 6.2.7  `search` Property

An optional set of search criteria for the target report for a drill-though link.

**Summary**

Display Name: Search Criteria

ROM Type: List of Search Criteria structures

Default value: None

Settable at runtime: Yes

Availability: After the first release

**Description**

Optional list of search criteria. The report is run (if needed) or an existing report is opened. Then, BIRT performs a search on the target report based on the search criteria entered here.

For example, if the current report is a list of customers, the report can include a hyperlink that links to a report that shows open orders for all customer, then search for the customer's orders. In this case, the search criteria look for the customer's ID within the target report.

## 6.3  Search Criteria Structure

Defines a search criteria for a drill-through hyperlink.

**Summary**

Display Name: Search Criteria

Availability: After the first release

**Properties**

`name`

The name of the field to search in the target report.

`op`

The search operation.

`expr`

The value for the search criteria.

**Description**

This element binds a data value in the report to a parameter to pass along with a request to view another document. Parameters are bound exactly. That is:

    *ParameterName = Expression*

    *ParameterName < Expression*

And so on.

### 6.3.1  `name` **Property**

The name of the field to search in the target report.

**Summary**

Display Name: Field Name

ROM Type: Name

Default value: None

Required.

Settable at runtime: Yes

Availability: After the first release

**Description**

The name of the field to search in the target report.

### 6.3.2   `op` Property

The search operation.

**Summary**

Display Name: Operator

ROM Type: Choice

JavaScript Type: String

Default value: = (equality)

Settable at runtime: Yes

Availability: After the first release

**Choices**

| Display Name | Internal Name | Description |
|---|---|---|
| Equals | = | Find values exactly equal to the expression. |
| Less Than | < | Find values less than the expression. |
| Less Than or Equal | <= | Find values less than or equal to the expression. |
| Greater Than or Equal | >= | Find values greater than or equal to the expression. |
| Greater Than | > | Find values greater than the expression. |
| Not Equal | != | Find values not equal to the expression. |

**Description**

Determines the kind of search to perform. The default is to perform an equality search.

### 6.3.3   `expr` Property

The value for which to search.

**Summary**

Display Name: Expression

ROM Type: Expression

Expression Type: String

Default value: None

Required.

Settable at runtime: Yes

Availability: Not in the first release

**Description**

The value for which to search.

## 6.4  Parameter Binding Structure

Represents an association between an expression and a data set or nested report input parameter.

**Summary**

Display Name: Parameter Binding

Availability: First release

**Properties**

`name`

    The name of an input parameter in the data set.

`expr`

    An expression used to compute the value of the parameter.

**Description**

The bindings allow a report item to pass data into its data set. For example, a report design may have a customer query from a personal database. For each customer, it wants to create a table of open orders from a sales database. The bindings allow the inner table item to pass values into the input parameters of the inner data set.

**See Also**

Report Item `bindings` Property

### 6.4.1  `name` Property

The name of an input parameter in the data set.

**Summary**

Display Name: Parameter Name

ROM Type: Name

JavaScript Type: String

Required.

Settable at runtime: No

Availability: First release

**Description**

This property identifies an input parameter within a data set or included report. The name must match a parameter defined in the data set or nested report.

**See Also**

Data Set `inputParams` property

Include element (not in the first release)

### 6.4.2  `expr` Property

An expression used to compute the value of the parameter.

**Summary**

Display Name: Value Expression

Type: Expression

ROM Expression type: Any

JavaScript Type: any

Default value: None

Required.

Availability: First release

**Description**

Provides the value to bind to the input parameter. The expression executes in Factory context. The value of the expression must be legal for the associated input parameter, or a runtime error may occur. If the expression refers to a data row, the row is that of the outer report item (if any). No data row is available yet for this report item (because the data set has not yet been opened.)

## Appendix A: Elements that Support Custom Properties

The following table shows which elements support custom properties.

| Element | Custom Property Support |
| --- | --- |
| Report Design | Not supported |
| Parameters (all types) | Supported |
| Data sources | Supported |
| Data sets | Supported |
| Styles | Not supported in first open source release. (Custom properties will require extensive code for support in styles, something that will be viable with the first commercial release, but not the first open source release.) |
| Master page | Supported |
| Report items | Supported |
| Translations | Not supported |
| Images | Not supported |

## Appendix B: Containment

The following table shows which elements can appear inside other elements. The first column lists "container" elements: those that can contain other elements. Container elements define one or more "slots". For tables and lists, the set of items that can appear in the column header slot differs from that which can appear in other slots.

| Content Element | Body | List | Table H | Table D, F, GH, GF | Matrix | Grid | Free-form (See note) | Comp-onents | Scratch Pad |
|---|---|---|---|---|---|---|---|---|---|
| Text | ✓ | ✓ | C | C | C | C | ✓ | ✓ | ✓ |
| Grid | ✓ | ✓ | C | C | C | C | ✓ | ✓ | ✓ |
| Free-form | ✓ | ✓ | C | C | C | C | ✓ | ✓ | ✓ |
| List | ✓ | ✓ | — | C | — | C | ✓ | ✓ | ✓ |
| Table | ✓ | ✓ | — | C | — | C | ✓ | ✓ | ✓ |
| Chart | ✓ | ✓ | — | C | C | C | ✓ | ✓ | ✓ |
| Dashboard item | ✓ | ✓ | | C | C | C | ✓ | ✓ | ✓ |
| Matrix | ✓ | ✓ | — | C | — | C | ✓ | ✓ | ✓ |
| Include | ✓ | ✓ | — | C | — | C | ✓ | ✓ | ✓ |
| TOC | ✓ | | — | — | — | | | ✓ | ✓ |

| Content Element | Body | List | Table H | Table D, F, GH, GF | Matrix | Grid | Free-form (See note) | Comp- onents | Scratch Pad |
|---|---|---|---|---|---|---|---|---|---|
| Label | ✓ | ✓ | C | C | C | C | ✓ | ✓ | ✓ |
| Data | ✓ | ✓ | C | C | C | C | ✓ | ✓ | ✓ |
| Image | ✓ | ✓ | C | C | C | C | ✓ | ✓ | ✓ |
| Line | ✓ | ✓ | | | | C | ✓ | ✓ | ✓ |
| Rectangle | | | | | | | ✓ | ✓ | ✓ |
| Ext. Item | ✓ | ✓ | C | C | C | C | ✓ | ✓ | ✓ |
| Browser Control | ✓ | ✓ | C | C | C | C | ✓ | ✓ | ✓ |
| Cell | | | ✓ | ✓ | ✓ | ✓ | | | |

The set of items that can appear in a grid or container is further constrained by context. If the grid or container appears inside a table, list or matrix, then the set of allowed elements is limited by what can appear in the enclosing element. For example, if a grid appears in a list column header, then it cannot contain a list or table.

Legend:

| | |
|---|---|
| ✓ | The element is allowed. |
| C | This slot defines a cell. The element is allowed inside a cell. |
| — | This element is not allowed inside this slot, either as a direct or indirect content. |

## Appendix C: Style Properties

The following table identifies the style properties that apply to each report item.

*Note:* *The highlight below shows cells that need to be verified.*

| Style Property | Graphic Master Page | Simple Master Page | Data | Free-Form | Grid | Image | Label | List | Multi-line Data | Table | Text | TOC | Row, Col, Cell (3) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| background Attachment | TBD | N | N | TBD | N | N | N | ? | Y | ? | Y | TBD | ? |
| background Color | Y | Y | Y | Y | Y | N | Y | Y | Y | Y | Y | TBD | ? |
| background Image | TBD | N | N | TBD | N | N | N | ? | Y | ? | Y | TBD | ? |
| background PositionX | TBD | N | N | TBD | N | N | N |  | Y |  | Y | TBD | ? |
| background PositionY | TBD | N | N | TBD | N | N | N |  | Y |  | Y | TBD | ? |
| background Repeat | TBD | N | N | TBD | N | N | N | ? | Y | ? | Y | TBD | ? |
| border *Side*Color | N | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| border *Side*Style | N | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| border *Side*Width | N | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| canShrink | N |  | Y | Y | ? | ? | Y | N | Y | N | Y | N | ? |

| Style Property | Graphic Master Page | Simple Master Page | Data | Free-Form | Grid | Image | Label | List | Multi-line Data | Table | Text | TOC | Row, Col, Cell (3) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Color | C | C | Y | C | C | Y | N | C | Y | C | Y | Y | Y |
| dateTime Format | C | C | Y | C | C | N | N | C | N | C | N | N | C |
| display | N | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | N | N |
| fontFamily | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| fontSize | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| fontStyle | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| fontVariant | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| fontWeight | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| highlight | N | N | Y | C | C | N | N | C | N | C | N | N | C |
| letter Spacing | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| line Height | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| map | N | N | Y | C | C | N | N | C | N | C | N | N | C |
| marginSide | N | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| master Page | N | | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1) | N |
| number Format | C | C | Y | C | C | N | N | C | N | C | N | N | C |
| orphans | N | N | N | C | C | N | N | C | N | C | N | N | C |
| padding Side | N | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |

| Style Property | Graphic Master Page | Simple Master Page | Data | Free-Form | Grid | Image | Label | List | Multi-line Data | Table | Text | TOC | Row, Col, Cell (3) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pageBreak After | N | N | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1) | N |
| pageBreak Before | N | N | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1) | N |
| pageBreak Inside | N | N | N | N | N | N | N | (1) | N | (1) | N | N | N |
| showIf Blank | N | N | Y | Y | Y | Y | N | C | Y | C | Y | N | N |
| string Format | C | C | Y | C | C | N | N | C | N | C | N | N | C |
| textAlign | C | C | Y | C | C | N | Y | C | Y | C | Y | N | C |
| text Decoration | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| text Underline | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| text Overline | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| textLine Through | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| textIndent | N | N | N | C | C | N | N | C | Y | C | Y | N | C |
| text Transform | C | C | Y | C | C | N | Y | C | Y | C | Y | Y | C |
| vertical Align | C | C | (2) | C | C | (2) | (2) | C | (2) | C | (2) | N | Y |

| Style Property | Graphic Master Page | Simple Master Page | Data | Free-Form | Grid | Image | Label | List | Multi-line Data | Table | Text | TOC | Row, Col, Cell (3) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| white Space | C | C | Y | C | C | N | Y | C | Y | C | Y | N | C |
| widows | N | N | N | C | C | N | N | C | N | C | N | N | C |
| word Spacing | C | C | Y | C | C | N | Y | C | Y | C | Y | N | C |

Notes:

1. Takes effect only if the report item acts as a section; ignored if the item is inside a container.

2. Takes effect only if the report item appears inside a grid or table cell.

3. The use of certain style properties for columns is limited by some browsers, especially IE. The order of precedence of all the style properties (from highest to lowest): the content of the cell, cell, row, column, table.

Legend:

| | |
|---|---|
| Y | The style property applies to this item directly. |
| C | The style property does not apply to this item, but cascades to the item's contents. |
| N | The style property does not apply to this item. |
| (*n*) | The style property applies, but only in the context explained in the note. |