

BIRT API Change Control Document

Report Engine Public Interfaces

Last Updated: November 14, 2005

BIRT API Change Control Document	1
Abstraction	1
1. Introduction	2
2. Changed APIs:	2
2.1 Class ReportEngine	2
2.2 Parameter Handling Related Interfaces	3
2.2.1 Interface <i>IParameterDefnBase</i>	3
2.2.2 Interface <i>IScalarParameter</i>	4
2.2.3 Interface <i>IGetParameterDefnTask</i>	4
2.3 Pass Context and Other Objects to Engine	5
2.3.1 Interface <i>IEngineTask</i>	5
3. Added APIs:	6
3.1 Utility Classes/Interfaces	6
3.1.1 Class <i>ComponentID</i>	6
3.1.2 Class <i>InstanceID</i>	6
3.2 Data Extraction Related Interfaces	7
3.2.1 Interface <i>IDataExtractionTask</i>	7
3.2.2 Interface <i>IExtractionResults</i>	9
3.2.3 Interface <i>IDataIterator</i>	10
3.3 Report Document Related	11
3.3.1 Interface <i>IRunTask</i>	11
3.3.2 Interface <i>IPageHandler</i>	12
3.3.3 Interface <i>IReportDocument</i>	12
3.3.4 Interface <i>IReportArchive</i>	14
3.4 Presentation Engine Related	15
3.4.1 Interface <i>IRenderTask</i>	15
3.4.2 Interface <i>IDesignModifier</i>	17
3.5 TOC-Related	17
3.5.1 Class <i>TOCNode</i>	17
3.6 Extended Item-Related	18
3.6.1 Class <i>IReportItemPresentation</i>	18
3.6.2 Class <i>IReportItemGeneration</i>	19

Abstraction

This document tracks the change request to the public API for BIRT Report Engine. For each change request, the document will describe the new requirement, proposed solution and follow-up actions.

1. Introduction

The Report Engine interface allows Java developers to access report generation and rendering services. In Release 1.0, this interface allows developers to configure and instantiate a report engine, to retrieve report parameters, or to run and render reports. However, several essential reporting functionalities are not supported. For example, BIRT 1.0 does not support running a report to generate a report document; nor does it support rendering a report based on a report document. User can only request for the report in a nonpaginated mode, so no page-on-demand viewing is supported. Other advanced features that are not supported include TOC, data extraction, etc.

This document details the proposed changes to the Report Engine interface for the BIRT 2.0 release.

2. Changed APIs:

2.1 Class ReportEngine

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

Support creating new tasks, and opening report document.

Proposed Solution:

Add methods in `ReportEngine` to support these functionalities. These methods use other classes/interfaces defined in the Added APIs section.

Impact on Existing Code (Backward-compatibility):

None

API Description:

```
/**
 * creates a task to run a report to generate a report document
 *
 * @param reportRunnable the runnable report design object
 * @return a task that runs the report
 */
public IRunTask createRunTask( IReportRunnable reportRunnable );

/**
 * creates a task that renders the report to a specific output format.
 *
 * @param reportDocument a handle to an IReportDocument object
 * @return a task that renders a report to an output format
 */
public IRenderTask createRenderTask( IReportDocument reportDocument );

/**
 * opens a report document and returns an IReportDocument object, from
 * which further information can be retrieved.
 *
 * @param docArchiveName the report document name. report document is an
```

```

* archive in BIRT.
* @return A handle to the report document
* @throws EngineException thrown when the report document archive does
* not exist, or the file is not a valid report document
*/
public IReportDocument openReportDocument( String docArchiveName )
    throws EngineException;

/**
 * creates a task that allows data extraction from a report document
 *
 * @param reportDocument a handle to an IReportDocument object
 * @return a task that renders a report to an output format
 */
public IDataExtractionTask createDataExtractionTask( IReportDocument
reportDocument );

/**
 * @param reportRunnable the runnable report design object
 * @return a task for previewing parameter data
 */
public IDataPreviewTask createDataPreviewTask(IReportRunnable
reportRunnable)

```

2.2 Parameter Handling Related Interfaces

2.2.1 Interface IParameterDefnBase

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

Support cascading parameter group.

Proposed Solution:

Added one more constant to the interface

Impact on Existing Code (Backward-compatibility):

None

API Description:

The following constant is added:

```
public static final int CASCADING_PARAMETER_GROUP = 5;
```

Function

```
public int getParameterType( );
```

may now return this value.

2.2.2 Interface IScalarParameter

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

The current parameter handling interface allows retrieval of default values, selection list values given a report parameter. Since the report parameter is not always associated with a task, methods such as evaluateDefaults(), fillDynamicParameterValues() on IGetParameterDefnTask has to be called before retrieval of parameter values or parameter value selection list.

We now propose to only support getting default values, selection lists, etc. on the IGetParameterDefinitionTask object. Function

```
public ArrayList getSelectionList( );
```

is there fore deprecated in this interface.

2.2.3 Interface IGetParameterDefnTask

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

For the same reason described above, we are deprecating some methods in IGetParameterDefnTask and introducing some other methods in this interface.

API Description:

The following methods are deprecated:

```
public abstract void evaluateDefaults() throws EngineException;
```

The following methods are added to support dynamic parameter list and cascading parameters:

```
/**
 * get default value by parameter name
 *
 * @param paramName name of the parameter
 * @return default value for a parameter
 */
public Object getDefaultValue(String paramName);

/**
 * @return a parameter value selection object, from which a list
 * of parameter values and label values can be retrieved.
 */
public ArrayList getSelectionList( );

/**
 * evaluates the cascading parameter values for a parameter group.
 * this function may cause the runtime environment
 * to be set up, and a query to be run. This function has to be
```

```

* called before the getCascadingSelectionList function can be
* called.
*
* @param parameterGroupName - the cascading parameter group name
*/
    public void evaluateQuery( String parameterGroupName )

/**
*
* @param parameterGroupName - the cascading parameter group name
* @param groupKeyValues - the array of parameter values preceding the
* current parameter, i.e., if we have country, state and city in a cascading
* parameter group, passing in ["USA", "CA"] to retrieve cities.
* @return the selection list of the parameter to work on
*/
    public Collection getCascadingSelectionList ( String parameterGroupName,
Object[] groupKeyValues )

```

2.3 Pass Context and Other Objects to Engine

2.3.1 Interface IEngineTask

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

We would like to pass in application, session, request, or render context information to engine so that engine can pass such info to other components. Because the information could be diverse, the approach for doing so is to create a Map object which contains a number of named objects.

The EngineConfig class already has something in this line, i.e., function addScriptableObject. This function makes an object accessible through Script only, but does not pass the objects to other environments. In addition, the name is inadequate, because we want to pass in anything we want, and we want the object to be accessible through Java-based scripting too. With the new Map object, anything added to the map is automatically added for JavaScript access.

API Description:

The function

```
public abstract void setContext(Object context);
```

is removed from Engine API. It is replaced with

```
public abstract void setAppContext(Map context);
```

Notice that this technically may break customer code. However, based on internal Engine logic, this method in the past has mainly been used by BIRT to pass HTMLenderContext to engine, so the impact should be minimal.

3. Added APIs:

3.1 Utility Classes/Interfaces

3.1.1 Class ComponentID

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

To support various BIRT operations, BIRT needs a way to uniquely identify a report component.

Proposed Solution:

Add an ComponentID class. Each instance corresponds to a unique report component in a report design. A report component could be at a finer granularity than a report element, i.e., each group, column, row, and cell gets its own component ID.

The current thought is that the class could simply wrap around an integer.

API Description:

```
/**
 * a class that wraps around an identifier for a report component
 */
public class ComponentID {
    public int getID() {return componentID;}
}
```

3.1.2 Class InstanceID

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

To support various BIRT operations, BIRT needs a way to uniquely identify a report component instance.

Proposed Solution:

Add an InstanceID class that wraps around a report item instance.

API Description:

```
/**
 * a class that wraps around an identifier for a report element instance
 */
public class InstanceID {
    /**
     * returns the component id for the element given an instance ID
     */
    public ComponentID getComponentID();
}
```

```

/**
 * returns the instance number
 */
public int getInstanceNumber();

/**
 * returns the resultset number
 */
public int getResultSetNumber();

/**
 * returns the string representation of the instance ID
 */
public String toString();
}

```

3.2 Data Extraction Related Interfaces

3.2.1 Interface IDataExtractionTask

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

Support data extraction. See BPS32 for BIRT release 2

Proposed Solution:

Add a data extraction task to engine. Add several helper classes/interfaces that this interface may use.

API Description:

```

/**
 * an engine task that extracts data from a report. The task allows the return
 * of metadata and data from engine
 *
 * User first creates the task from engine, then sets a report component ID,
 * or report component instance ID. If none is set, data extraction is assumed
 * to be based on all the data stored in the report. The user
 * can call the getMetaData method to retrieve metadata for each resultset.
 * Based on the metadata, he can select additional columns, add filter
 * conditions, or specify sorting conditions.
 */
public interface IDataExtractionTask extends IEngineTask {

    public static int SORT_DIRECTION_ASCENDING = 0;
    public static int SORT_DIRECTION_DESCENDING = 1;

    /**
     * sets the report item identifier that data extraction will happen on
     *
     * @param cid report item identifier
     */
    public abstract void setItemID(ComponentID cid);

    /**

```

```

    * @param iid identifies a report item instance that data extraction
    * will happen on
    */
    public abstract void setInstanceID(InstanceID iid);

    /**
     * returns the metadata corresponding to the data stored in the report
     document, for the specific extraction level, i.e., report, data set, report
     item, or report item instance levels.
     * To get the metadata for the extracted data, use the
     getResultMetaData method from the
     * IDataIterator interface.
     *
     * @return a List of IResultMetaData. The list usually has one result
     set meta data, but
     * could have more if data extraction is based on the whole report
     */
    public abstract List getMetaData();

    /**
     * @param columnName name of the column to be included in the data set
     */
    public abstract void selectColumns(String[] columnNames);

    /**
     * @param simpleFilterExpression add one filter condition to the
     * extraction. Only simple filter expressions are supported for now,
     * i.e., LHS must be a column name, only <, >, =, <=, >=
     * and startWith is supported.
     */
    public abstract void setFilters(Filter[] simpleFilterExpression);

    /**
     * @param columnNames names of the columns to sort on
     * @param directions the directions for sorting the data based on the
     * specified columns
     */
    public abstract void setSortConditions(String[] columnNames, int[]
directions);

    /**
     * sets query string for data extraction. Not supported now.
     *
     * @param queryString a query string that acts as extraction criterion
     */
    // public abstract void setQuery(String queryString);

    /**
     * @return an object of type IExtractionResults, from which data
     iterators can be obtained and
     * data can be retrieved
     */
    public abstract IExtractionResults extract( ) throw EngineException;
}

```


3.2.2 Interface IExtractionResults

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

Support data extraction.

Proposed Solution:

This interface wraps around an extraction handle, from which user can extract data stored in multiple result set, together with the metadata. This interface is creates following IQueryResults in data engine.

API Description:

```
/**
 * A handle used to retrieve data stored in a report. Extraction results
 * could contain multiple resultsets, especially when the extraction is
 * at report level, i.e., get all the data stored for a report
 */
public interface IExtractionResults
{
    /**
     * Returns the metadata of the first or current result set <br>
     * This method provides the result metadata without having to
     * first fetch the result data. <p>
     * Returns Null if the metadata is not available before fetching
     * from an <code>IResultIterator</code>,
     * or if it is ambiguous on which result set to reference.
     * In such case, one should obtain the result metadata
     * from a specific <code>IResultIterator</code>.
     * @return The metadata of the first result set's detail row in this
     * <code>IQueryResults</code>. Null if not available or
     * ambiguous on which result set to reference.
     * @throws EngineException if error occurs during extraction
     */
    public IResultMetaData getResultMetaData() throws BirtException;

    /**
     * Returns the current result's iterator.
     * Repeated call of this method without having advanced to the next
     * result would return the same iterator at its current state.
     * @return The current result's iterator.
     * @throws EngineException if error occurs during extraction
     */
    public IDataIterator nextResultIterator() throws BirtException;

    /**
     * Closes all query result set(s) associated with this object;
     * provides a hint to the query that it can safely release
     * all associated resources.
     * The query results might have iterators open on them.
     * Iterators associated with the query result sets are invalidated
     * and can no longer be used.
     */
    public void close();
}
```

3.2.3 Interface IDataIterator

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

Support data extraction.

Proposed Solution:

An iterator class that allows data to be retrieved row-by-row. Only a `getValue()` method, which returns an `Object`. Convenience methods such as `getBoolean`, `getInteger` etc. may be added in the future.

This interface is created following `IQueryIterator` interface in data engine.

API Description:

```
/**
 * An iterator on a result set from a prepared and executed query.
 * Multiple IResultIterator objects could be associated with the same
 * IExtractionResults object, if extraction is done at report level
 */
public interface IDataIterator
{
    /**
     * Returns {@link org.eclipse.birt.report.engine.api.IExtractionResults}
     * from which this data iterator is obtained.
     */
    public IExtractionResults getQueryResults();

    /**
     * Returns the metadata of this result set's detail row.
     * @return The result metadata of a detail row.
     */
    public IResultMetaData getResultMetaData() throws BirtException;

    /**
     * Moves down one element from its current position of the iterator.
     * @return true if next element exists and
     * has not reached the limit on the maximum number of rows
     * that can be accessed.
     * @throws BirtException if error occurs
     */
    public boolean next() throws BirtException;

    /**
     * Returns the value of a column.
     * @param columnName the name of the column
     * @return The value of the given column. It could be null.
     * @throws BirtException if error occurs
     */
    public Object getValue( String columnName ) throws BirtException;

    /**
     * @param index column index. It is 1 based
     * @return The value of the given column. It could be null.
     * @throws BirtException if error occurs
     */
    public Object getValue( int index ) throws BirtException;
}
```

```

/**
 * Closes this result and provide a hint that the consumer is done with
 * this result,
 * whose resources can be safely released as appropriate.
 */
public void close();
}

```

3.3 Report Document Related

3.3.1 Interface IRunTask

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

Support running a report to generate report document.

Proposed Solution:

A task for running a report to generate report document. Most functions listed exist in the BIRT 1.0 IRunAndRenderTask.

API Description:

```

/**
 * An engine task that runs a report and generates a report document.
 */
public interface IRunTask extends IEngineTask {

    /**
     * set all parameter values
     * @param params a hash map with all parameters
     */
    public abstract void setParameterValues(HashMap params);

    /**
     * sets one parameter value
     * @param name parameter name
     * @param value parameter value
     */
    public abstract void setParameterValue(String name, Object value);

    /**
     * returns the parameter name/value collection
     * @return the parameter names/values in a hash map
     */
    public abstract HashMap getParameterValues();

    /**
     * @return whether the parameter validation succeeds <br>
     */
    public boolean validateParameters( );

    /**
     * set up event handler to be called after each page is generated
     *
     * @param callback a callback function that is called after each

```

```

    * checkpoint
    */
    public void setPageHandler(IPageHandler callback);

    /**
     * runs the task to generate report document
     * @param manager an interface for writing to / reading from disk when
     * generating report document
     * @param reportDocArchiveName the name for the report document file
     * @throws EngineException throws exception when running report fails
     */
    public abstract void run(IReportDocManager manager, String
reportDocName) throws EngineException;
}

```

3.3.2 Interface IPageHandler

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = Application Developer

Change Request:

Allows customization code after each page is generated. May be used in progressive viewing.

Proposed Solution:

An interface that allows the application developer to define event handler that is called on each page generated.

API Description:

```

/**
 * An interface implemented by app developer to provide handler after each
 * page is generated in factory. Can be used to support checkpointing, and
 * therefore progressive viewing.
 */
public interface IPageHandler {
    /**
     * @param doc the report document
     * @param pageNumber page indexed by pageNumber has finished generation
     * @param checkpoint whether the page indexed by pageNumber is ready for
     * viewing
     */
    public void onPage(IReportDocument doc, int pageNumber, boolean
checkpoint);
}

```

3.3.3 Interface IReportDocument

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

The high-level report document interface. Functionalities limited to retrieval of high-level information stored in the report only, i.e., no runtime support is needed. Data extraction functionalities are supported through the data extraction task interface.

Proposed Solution:

An interface that allows the application developer to retrieve stored information from the report document.

API Description:

```
/**
 * A report document (i.e., not modifiable) that can be rendered to
 * other formats in the BIRT presentation engine
 *
 * This is the high-level report document interface. Low-level report document
 * interface that supports file-level operations is in a separate interface.
 */
public interface IReportDocument {

    /**
     * @return the report document (archive) name
     */
    public abstract String getReportDocumentName();

    /**
     * @return a report design stream. This is useful for rerunning a report
     * based on report document
     */
    public abstract InputStream getDesignStream();

    /**
     * @return the runnable report design. It is available because a report
     * document must be run
     * with a report design
     */
    public abstract IReportRunnable getReportRunnable();

    /**
     * returns values for all the parameters that are used for generating
     * the current report document. Useful for running the report again
     * based on a report document
     *
     * @return parameter name/value pairs for generating the current report
     * document.
     */
    public abstract HashMap getParameterValues();

    /**
     * @return the page count in the report. Used for supporting page-based
     * viewing
     */
    public abstract int getPageCount();

    /**
     * Given a report item instance idD, returns the page number
     * that the instance starts on (to support Reportlet).
     *
     * @param iid report item instance id
     * @return the page number that the instance appears first
     */
    public abstract int getPageNumber( InstanceID iid );

    /**
     * Given a bookmark in a report, find the (first) page that the
     * bookmark appears in (for hyperlinks to a bookmark)
```

```

    * @param bookmarkName bookmark name
    * @return the page number that the instance appears first
    */
    public abstract int getPageNumber( String bookmark );

    /**
     * @return a list of bookmark strings
     */
    public abstract List getBookmarks();

    /**
     * @param parent id of a TOC node. Pass null or "" as the root TOC node
     * @return A list of TOC nodes that are direct child of the parent
     * node
     */
    public abstract List getTOC( String parentNodeID );
}

```

3.3.4 Interface IReportArchive

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

DtE and/or third-party components may want to write report documents. They need an interface from engine to create report archive or reading from/write to the archive.

Proposed Solution:

The file-level interface for reading/writing a report archive. Provides services for writer components to get stream or storages. Actual reading/writing the report document streams are still through File APIs.

API Description:

```

    /**
     * An interface that wraps around a report archive. A report archive
     * is a zip file in compressed format, a folder in uncompressed
     * format.
     *
     * Notice that the interface does not define archive file name, nor
     * does it define folder name to store/uncompress the archive to.
     * Setting such environments up is implementation class's
     * responsibility. To external users of IReportArchive, it only
     * cares what he can retrieve from the archive.
     */
    public interface IReportArchive {

        /**
         * sends the report document to the output stream. If the report
         * archive is not compressed yet, it is compressed first.
         *
         * @param ostream an output stream to send the report archive (the
         * compressed report doc) to
         */
        public abstract void writeToStream(OutputStream ostream);

    }

```

```

    * creates a compressed version of the report document. Usually
    * called after all the streams in the report document are
    * generated
    */
    public void compress();

    /**
     * decompresses a report item
     */
    public void decompress();

    /**
     * returns a sequential access file. The path is based on Unix
     * syntax, with the root of the archive denoted by "/". The
     * initial "/" character can be skipped.
     * Used mainly for sequential streams in report.
     *
     * @param fullPath the full path to the file
     * @param mode the mode to create
     * @return a File object for the specific stream
     */
    public File getStream(String fullPath, String mode);

    /**
     * returns a random access file. The path is based on Unix syntax,
     * with the root of the archive denoted by "/". The initial "/"
     * character can be skipped. That
     * is, "/data" and "data" both means the data stream. Used mainly
     * for random access streams such as search indices, etc.
     *
     * @return a random access file object
     */
    public RandomAccessFile getRandomAccessStream(String fullPath,
String mode);

    /**
     * @param fullPath the full path to the file
     * @return whether the stream exist
     */
    public boolean hasStream(String fullPath);

    /**
     * @param path the path for a storage, or stream.
     * @return a list of strings representing the underlying stream
     * names. If a stream
     * path is passed to the function, returns null.
     */
    public List listStreams(String path);
}

```

3.4 Presentation Engine Related

3.4.1 Interface IRenderTask

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

Support rendering a report to different formats basd on a report document

Proposed Solution:

A task for rendering a report/a report item/a page of a report to specified output format.

API Description:

```
/**
 * An engine task that renders a Report Document to one of the output formats
 * supported by the engine.
 */
public interface IRenderTask extends IEngineTask {

    /**
     * set the rendering options
     * @param settings the rendering options
     */
    public abstract void setRenderOption(IRenderOption options);

    /**
     * @return the render option
     */
    public abstract IRenderOption getRenderOption();

    /**
     * set the callback function that the engine calls to allow the caller
     * to have a final change on the report design before running the report
     * @param dm design modifier, which allows the design to be modified
     * before running the report. Not supported in BIRT 2.0
     */
    // public abstract void setDesignModifier(IDesignModifier dm);

    /**
     * sets a specific emitter to use when generate output. Used when there
     * are more than one emitters that support a single format. One example
     * is the FO-based PDF emitter and the new PDF emitter added in BIRT2.0.
     * If this function is not called when there are more than 1 emitters
     * that support a format, engine may arbitrarily pick one.
     * @param id the identifier for the emitter
     */
    public abstract void setEmitterID(String id);

    /**
     * render the whole report document or an output format
     *
     * @throws EngineException if rendering fails
     */
    public abstract void render() throws EngineException;

    /**
     * @param pageNumber
     * @throws EngineException
     */
    public abstract void render(int pageNumber) throws EngineException;

    /**
     * Render the page from startPageNumber to endPageNumber in the Report
     * Document to an output format. Not supported in BIRT 2.0.
     * @throws EngineException
     */
    //public abstract void render(String pageRange) throws EngineException;
```



```

/**
 * Render the ReportLet whose container is identified by iid.
 * Useful for Reportlet support. Not supported in BIRT 2.0.
 *
 * @param itemInstanceID the report iteminstance to be rendered
 * @throws EngineException
 */
//public abstract void render( InstanceID iid ) throws EngineException;
}

```

3.4.2 Interface IDesignModifier

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = Host of Engine

Change Request:

Support rendering a report with user changes from interactive viewing or web reporting. Not supported in BIRT 2.0.

Proposed Solution:

Add an interface so that the caller can define a callback function.

API Description:

```

/**
 * An engine task that renders a Report Document to one of the output formats
 * supported by the engine.
 */
public interface IDesignModifier {

    /**
     * allows modification to the report design before rendering it
     * @param handle the report design handle
     */
    public abstract void modify(IReportDesignHandle handle);
}

```

3.5 TOC-Related

3.5.1 Class TOCNode

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.api

Implementer = BIRT Team

Change Request:

Support TOC.

Proposed Solution:

The IReportDocument interface already supports a getChildList method that is used for TOC. This class defines one TOCNode, which is used in the IReportDocument interface.

API Description:

```

/**

```

```

* A node that wraps around a TOC entry
*/
public class TOCNode {
    protected String displayString;

    /**
     * @return returns the parent node of the current TOC node
     */
    public TOCNode getParent();

    /**
     * @return the list of child TOC nodes
     */
    public List getChildren();

    /**
     * @return the display string for the TOC entry
     */
    public String getDisplayString();

    /**
     * @return the node id (bookmark string) that the TOC node points to.
     */
    public String getNodeID();

    /**
     * @return the bookmark string that the TOC item points to.
     */
    public String getBookmark();
}

```

3.6 Extended Item-Related

3.6.1 Class IReportItemPresentation

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.extension

Implementer = Extension Writer

Change Request:

1. BIRT now supports Image map as another type that an extension can generate.
2. Pass scripting context object to cart
3. Pass a unique ID for the extended item to extension writer. The unique id passed to extension item is unique within the report document.

Proposed Solution:

Add one constant in this interface for supporting image map.

Add two setter functions for setting context object and instance ID.

API Description:

The following constant is added:

```
public static int OUTPUT_AS_IMAGE_WITH_MAP = 6;
```

The following two functions are added

```
// returns an identifier for the extended item instance  
public abstract void setID(String ID);
```

```
// returns an identifier for the extended item instance  
public abstract void setScriptContext(Object context);
```

3.6.2 Class IReportItemGeneration

Component name = Report Engine Interfaces

Package name = org.eclipse.birt.report.engine.extension

Implementer = Extension Writer

Change Request:

Similar to the IReportItemPresentation interface changes, we need to

1. Pass scripting context object to cart
2. Pass a unique ID for the extended item to extension writer. The unique id passed to extension item is unique within the report document.

Proposed Solution:

Add two setter functions for setting context object and instance ID.

API Description:

The following two functions are added

```
// returns an identifier for the extended item instance  
public abstract void setID(String ID);
```

```
// returns an identifier for the extended item instance  
public abstract void setScriptContext(Object context);
```