Diploma Thesis

CPC
an Eclipse framework for automated clone life cycle tracking and update anomaly detection

CPC Core API Specification

Valentin Weckerle
Freie Universität Berlin

January 23, 2008

Contents

Chapter 1

Package classifier.api.strategy

Package Contents *Page*

1.1 Interfaces

1.1.1 Interface **IClassificationStrategy**

API interface for new strategies which want to plug into the `CPC Classifier's IClassificationProvider implementation.`

Declaration

```
public interface IClassificationStrategy
```

Fields

- `public static final String CLASSIFICATION_REJECT`
  - `A special classification which indicates whether the clone should be rejected.`

Methods

- *classify*
  `public IClassificationStrategy.Status` **classify**`( IClassificationProvider.Type` **type**`, ICloneFile` **cloneFile**`, IClone` **clone**`, String` **fileContent**`, IClone` **originClone**`, Map` **result** `)`
  - **Usage**

    * Takes a clone object and the content of the file which contains the clone and tries to find good classifications for the clone.

      A strategy will add its results to the given result map.
      The clone object itself is **not** modified in any way.
      Keys of the result map are classification strings as defined in IClassificationProvider or custom strings defined by 3rd party plugins.
      The values specify the weight of the corresponding classification if an incremental type is selected, all old classifications are initially added with weight 1.0 to the result map.
      Each strategy may increase or decrease the values for any classification, according to its own judgement of the clone.
      After all strategies have been applied, the IClassificationProvider will keep all classifications with a weight >0.

      The key #CLASSIFICATION_REJECT ("cpc.reject") is a special case. Its value determines whether the clone will be accepted or rejected. The default value is 0.
      If the value is >0 after all strategies have been executed, the clone is rejected.
  - **Parameters**
    * type - IClassificationProvider.Type classification type, never null.
    * cloneFile - the clone file which contains the clone, must not be modified, never null.
    * clone - the clone to classify, must not be modified, never null.
    * fileContent - the content of the corresponding file, never null.
    * originClone - optional origin clone, may be NULL.
    * result - a result map with all classifications and their weight, a strategy writes its results to this map, never null.
  - **Returns** - the Status of the strategy execution, never null.

1.2 Classes

1.2.1 Class **IClassificationStrategy.Status**

Return status indicator for `#classify(IClassificationProvider.Type, ICloneFile, IClone, String, IClone, Map)` .

Declaration

public static final class IClassificationStrategy.Status
extends Enum

Fields

- public static final IClassificationStrategy.Status SKIPPED
  - Indicates that the strategy does not apply to the given clone and did not make any modifications.
- public static final IClassificationStrategy.Status MODIFIED
  - Indicates that the strategy made some modifications to the classification of the clone.
- public static final IClassificationStrategy.Status BREAK
  - Indicates that this event should not be passed on to any more strategies and that the clone's classification is in it's final stage.

    A strategy will typically return this value if it detected a special situation which may confuse other strategies or if it needs to make sure that no other strategy will override its decision.

Chapter 2

Package core.api.data

Package Contents *Page*

2.1 Interfaces

2.1.1 Interface **IClone**

Public interface for all clone data objects.

This interface lists all methods which are available to all CPC plug-ins and 3rd party contributions.

Additional methods are defined by more specific sub-interfaces which belong to individual CPC plugins and are to be considered private.
Any CPC plugin other than the one designated in the sub-interface API must not access such methods.

Any implementation needs to implement `ICloneInterfaces` . Implementing only `IClone` is not enough!

Declaration

public interface IClone
implements Comparable, ICloneObject

Fields

- public static final String PERSISTENCE_CLASS_IDENTIFIER
  - `IStatefulObject` persistence class identifier, value: "`clone`"

Methods

- *addClassification*
  `public void` **addClassification**`( String` **classification** `)`
  - **Usage**
    - Adds the given classification string to this clone.
      Multiple additions of the same string have no effect.

      Refer to the `CLASSIFICATION_*` constants of the `IClassificationProvider` for more information.
  - **Parameters**
    - `classification` - the classification string to add, never null.
  - **See Also**
    - `IClassificationProvider`

- *compareTo*
  `public int` **compareTo**`( IClone` **o** `)`
  - **Usage**
    - This is a somewhat tricky implementation of compareTo().
      By contract `this.compareTo(o) == 0` must always yield the same result as `this.equals(o)`.

      However, we're ordering by start line, start offset, end offset here.
      Two clones which are not equal may well start at the same line/offset.
      If this happens some extra code tries to resolve the issue by putting one of the clones first and the other one second.

- *getClassifications*
  `public Collection` **getClassifications**`( )`
  - **Usage**
    - Returns a collection with all classifications of this clone.
      All elements of the collection are unique, there are no NULL elements.

      NOTE: The returned collection may not be modified in any way. It may or may not be backed by the internal classification data structure. A client who wants to iterate over the set while the clone might be concurrently modified, should create its own shallow copy.

      Refer to the `CLASSIFICATION_*` constants in `IClassificationProvider` for more information.
  - **Returns** - classification of this clone, collection must not be modified, never null.
  - **See Also**

    - `IClassificationProvider`

- *getCloneState*
  `public IClone.State` **`getCloneState`**`( )`
  - **Usage**
    - Retrieves the `State` of this clone instance.
  - **Returns** - current state of this clone, never null.

- *getCloneStateChangeDate*
  `public Date` **`getCloneStateChangeDate`**`( )`
  - **Usage**
    - Retrieves the date of the last modification to this clone's `State` .
      This value is automatically updated to the current time, whenever the clone state is modified.
      Initially this value matches the `IClone#getCreationDate()` .
  - **Returns** - date of last state change, never null.
  - **See Also**
    - `IClone.setCloneState(State, double, String)`

- *getCloneStateDismissalDate*
  `public Date` **`getCloneStateDismissalDate`**`( )`
  - **Usage**
    - Retrieves the date of the last dismissal of a cpc notification for this clone by the user.
      The date can be used to retrieve the clone content from the history at the point in time where the notification was dismissed.
      In most cases this value will be NULL.
  - **Returns** - date of last dismissal of a cpc notification, may be NULL.

- *getCloneStateMessage*
  `public String` **`getCloneStateMessage`**`( )`
  - **Usage**
    - Retrieves an optional message which contains the rationale for the current clone state of this clone.
      This value is only defined for the states `State#NOTIFY` and `State#WARN` . However, the value is optional and can be NULL at any time.
      For all other states the value is always NULL.

      This value is displayed to the user and should therefore be human readable and localised.
  - **Returns** - human readable reason behind the current clone state, this value may be NULL if no reason was given or if the clone state is neither `State#NOTIFY` nor `State#WARN` .

- *getCloneStateWeight*
  `public double` **`getCloneStateWeight`**`( )`
  - **Usage**
    - Retrieves the weight of the current clone state of this clone.
      This value only has a meaning for the states `State#NOTIFY` and `State#WARN` . Otherwise the value is 0.
  - **Returns** - the weight of the current clone state, 0 if the state is neither `State#NOTIFY` nor `State#WARN` .
  - **See Also**
    - `IEvaluationResult.getWeight()`

- *getContent*
  `public String` **`getContent`**`( )`
  - **Usage**

    - ∗ Retrieves the current content of this clone.
      Calling this method may be expensive as contents may be lazy loaded.
  - **Returns** - current content of clone, never null.

- *getCreationDate*
  `public Date` **getCreationDate**`( )`
  - **Usage**
    - ∗ Retrieves the creation date of this clone.
  - **Returns** - creation date, never null.

- *getCreator*
  `public String` **getCreator**`( )`
  - **Usage**
    - ∗ Retrieves the creator (username) of this clone.
      This value may be null if the creator could not be determined.
  - **Returns** - creator of this clone, may be NULL.

- *getEndOffset*
  `public int` **getEndOffset**`( )`
  - **Usage**
    - ∗ Returns the offset of the last character which is still part of this clone.
      Use `IClone#setOffset(int)` and `IClone#setLength(int)` to modify this value.
      Convenience method.
  - **Returns** - offset + length - 1

- *getFileUuid*
  `public String` **getFileUuid**`( )`
  - **Usage**
    - ∗ Retrieves the UUID for the clone file in which this clone is located.
  - **Returns** - clone file uuid, never null.

- *getGroupUuid*
  `public String` **getGroupUuid**`( )`
  - **Usage**
    - ∗ Retrieves the UUID of the clone group which this clone belongs to.
      Initially a clone belongs to no group, in which case this value is NULL.
      As long as a clone is a member of a clone group, this value is non-NULL.
  - **Returns** - clone group for this clone, if any, may be NULL

- *getLength*
  `public int` **getLength**`( )`
  - **Usage**
    - ∗ Retrieves the length of this clone.
      A clone can't have length 0.
  - **Returns** - length in characters, never <=0.

- *getModificationDate*
  `public Date` **getModificationDate**`( )`
  - **Usage**

    - Retrieves the date of the last modification to this clone's content.
      Initially this value matches the `IClone#getCreationDate()` .
  - **Returns** - modification date, never null.

- *getOffset*
  `public int` **getOffset**`( )`
  - **Usage**
    - Retrieves the offset of the first character which is part of this clone.
  - **Returns** - the character offset to the beginning of the file at which the clone begins, inclusive, first char is 0.

- *getOriginalContent*
  `public String` **getOriginalContent**`( )`
  - **Usage**
    - Retrieves the original content of this clone at the time of its creation.
      Calling this method may be expensive as contents may be lazy loaded.
  - **Returns** - original content of clone, never null.

- *getOriginUuid*
  `public String` **getOriginUuid**`( )`
  - **Usage**
    - Retrieves the origin clone from which this clone was copied.
      May be null if this clone has no origin (i.e. it is only the source for other clones).
      A clone's origin may also be deleted, in which case this value is reset to NULL.
  - **Returns** - uuid of the origin clone of this clone or NULL if no origin exists.

- *hasClassification*
  `public boolean` **hasClassification**`( String` **classification** `)`
  - **Usage**
    - Checks whether this clone possesses the given classification.

      Refer to the `CLASSIFICATION_*` constants of the `IClassificationProvider` for more information.
  - **Parameters**
    - `classification` - the classification to check for, never null.
  - **Returns** - true if the clone has that classification, false otherwise.
  - **See Also**
    - `IClassificationProvider`

- *intersects*
  `public boolean` **intersects**`( IClone` **clone** `)`
  - **Usage**
    - Checks whether two clone positions intersect.
      Convenience method.
  - **Parameters**
    - `clone` - the other clone to compare against, never null.
  - **Returns** - true if the two position ranges have at least one character in common, false otherwise.

- *intersects*
  `public boolean` **intersects**`( int` **offset**`, int` **length** `)`
  - **Usage**

    * Checks whether this clone intersect with the given range.
      Convenience method.
  - **Parameters**
    * `offset` - start offset, 0-based character count, always >= 0.
    * `length` - length in characters, always >= 0. A length of 0 is handled like a length of 1 (endOffset=offset). Which means a 0-length range may intersect with another range.
  - **Returns** - true if the two position ranges have at least one character in common, false otherwise.

- *isTransient*
  `public boolean` **isTransient**`( )`
  - **Usage**
    * Whether this clone instance should be persisted or not.
      `IClone` instances may be created for temporary use, i.e. to keep track of the source for CutCopyPaste actions. Such transient `IClone` instances must be tracked like normal instances (their position can change due to modifications to the file), however they are not yet part of any clone group and are therefore not real clones. Such transient instances must not be persisted.
  - **Returns** - `true` if this clone should not yet be persisted, `false` otherwise.

- *removeClassification*
  `public void` **removeClassification**`( String` **classification** `)`
  - **Usage**
    * Removes the given classification from this clone.
      Has no effect if the clone did not possess the classification.

      Refer to the `CLASSIFICATION_*` constants of the `IClassificationProvider` for more information.
  - **Parameters**
    * `classification` - the classification string to remove, never null.
  - **See Also**
    * `IClassificationProvider`

- *setCloneState*
  `public void` **setCloneState**`( IClone.State` **cloneState**`, double` **weight**`, String` **message** `)`
  - **Usage**
    * Sets the `State` of this clone instance.
      A call to this method will also update `#getCloneStateChangeDate()` .
  - **Parameters**
    * `cloneState` - new state for this clone, never null.
    * `weight` - weight of the new state. This value only has a meaning for states `State#NOTIFY` and `State#WARN` and should be 0 for all others.
    * `message` - optional human readable reason behind the new clone state. This value should only be defined for states `State#NOTIFY` and `State#WARN` (even then it may be NULL) and should be NULL for all other states.
  - **See Also**
    * `IClone.getCloneState()`
    * `IClone.getCloneStateChangeDate()`
    * `IClone.getCloneStateWeight()`
    * `IClone.getCloneStateMessage()`

- *setGroupUuid*
  `public void` **setGroupUuid**`( String` **groupUuid** `)`
  - **Usage**

    - Sets the UUID of the clone group which this clone belongs to.
  - **Parameters**
    - `groupUuid` - the clone group for this clone, if any, may be NULL
  - **See Also**
    - `IClone.getGroupUuid()`

- *setLength*
  `public void` **setLength**`( int` **length** `)`
  - **Usage**
    - Sets the length of this clone.
  - **Parameters**
    - `length` - the length in characters, never <=0.
  - **See Also**
    - `IClone.getLength()`

- *setOffset*
  `public void` **setOffset**`( int` **offset** `)`
  - **Usage**
    - Sets the offset of the first character which is part of this clone.
  - **Parameters**
    - `offset` - the character offset to the beginning of the file at which the clone begins, inclusive, first char is 0.
  - **See Also**
    - `IClone.getOffset()`

- *setOriginUuid*
  `public void` **setOriginUuid**`( String` **originUuid** `)`
  - **Usage**
    - Sets the origin clone for this clone.
      This method is usually only used during the creation of the clone but may be used again at a later point to reset the origin uuid to NULL in case the origin clone was deleted.
  - **Parameters**
    - `originUuid` - uuid of the origin clone, may be NULL.

- *setTransient*
  `public void` **setTransient**`( boolean` **_transient** `)`
  - **Usage**
    - Specifies whether this clone instance should be persisted or not.
  - **Parameters**
    - `_transient` - `true` if this clone should not yet be persisted, `false` otherwise.
  - **See Also**
    - `IClone.isTransient()`

2.1.2 Interface **ICloneDataElement**

Root interface of all CPC data objects.

Each CPC data object is either one of these types:
- `ICloneObject`
- `ICloneObjectSupport`
- `ICloneObjectExtension`

Declaration

```
public interface ICloneDataElement
```

Methods

- *isSealed*
  `public boolean` **`isSealed`**`( )`
  - **Usage**
    - Checks whether this `ICloneDataElement` instance has been sealed.
  - **Returns** - true if this instance has been sealed.

- *seal*
  `public void` **`seal`**`( )`
  - **Usage**
    - Seals this `ICloneDataElement` instance.
      A sealed object may not be modified in any way. Otherwise an IllegalStateException is thrown.
      Sealing an already sealed object has no effect.

      In order to "unseal" an `ICloneDataElement` , it needs to be cloned. The seal state will not be propagated to the cloned instance.

2.1.3 Interface **ICloneFile**

Public interface for all clone file data objects.

Any implementation needs to implement `ICloneFileInterfaces` . Implementing only `ICloneFile` is not enough!

This interface lists all methods which are available to all CPC plugins and 3rd party contributions.

Additional methods are defined by more specific sub-interfaces which belong to individual CPC plugins and are to be considered private.
Any CPC plugin other than the one designated in the sub-interface API must not access such methods.

Declaration

```
public interface ICloneFile
implements ICloneObject
```

Fields

- public static final String PERSISTENCE_CLASS_IDENTIFIER
  - `IStatefulObject` persistence class identifier, value: "`clone_file`"

METHODS

- *getModificationDate*
  `public long` **`getModificationDate`**`( )`
  - **Usage**
    - Retrieves the file modification timestamp at the point in time of the last save operation. This value may not always be available.
  - **Returns** - the file modification timestamp of the corresponding file on disk in bytes.

- *getPath*
  `public String` **`getPath`**`( )`
  - **Usage**
    - Retrieves the project relative path of this file.
  - **Returns** - path of resource, relative to project directory. Contains the file name, but not the project name. Never null.

- *getProject*
  `public String` **`getProject`**`( )`
  - **Usage**
    - Retrieves the project name of this file.
  - **Returns** - project which contains the resource, never null.

- *getSize*
  `public long` **`getSize`**`( )`
  - **Usage**
    - Retrieves the file size at the point in time of the last save operation.
  - **Returns** - the file size of the corresponding file on disk in bytes.

2.1.4 INTERFACE **ICloneGroup**

Public interface for all clone group data objects.

Any implementation needs to implement `ICloneGroupInterfaces` . Implementing only `ICloneGroup` is not enough!

This interface lists all methods which are available to all CPC plugins and 3rd party contributions.

Additional methods are defined by more specific sub-interfaces which belong to individual CPC plugins and are to be considered private.
Any CPC plugin other than the one designated in the sub-interface API must not access such methods.

IMPORTANT NOTE

In order to ease the implementation of file based local storage providers as well as repository based remote store providers clone group objects are **not persisted individually** (clone group uuids are stored together with each clone member, nothing else is persisted).

An clone group object only contains it's uuid as a unique identifier and cached/derived counters and statistics.

A custom clone group implementation **must not** expect any data fields to be persisted.

This also means that `ICloneObjectExtensionStatefulObject` s will **not** be persisted.

Declaration

public interface ICloneGroup
implements ICloneObject

Fields

- public static final String PERSISTENCE_CLASS_IDENTIFIER
  - `IStatefulObject` persistence class identifier, value: "`clone_group`"

2.1.5 Interface **ICloneObject**

Base interface for all CPC Data Objects.

Any implementation needs to implement `ICloneObjectInterfaces` . Implementing only `ICloneObject` is not enough!

This interface lists all methods which are available to all CPC plugins and 3rd party contributions.

Additional methods are defined by more specific sub-interfaces (in the `*.api.data.special` package) which belong to individual CPC plugins and are to be considered private.
Any CPC plugin other than the one designated in the sub-interface API must not access such methods.
This does not apply to the main sub-interfaces in the `*.api.data` package.

General Considerations valid for all types of CPC Data Objects:
- All implementations must be **serializable**, **cloneable** and **adaptable**.
- All implementations must provide a zero argument constructor which auto generates a `uuid` as well as a constructor which takes one `String uuid` argument.

Declaration

public interface ICloneObject
implements ICloneDataElement, org.eclipse.core.runtime.IAdaptable, java.io.Serializable, Cloneable

Fields

- public static final String PERSISTENCE_OBJECT_IDENTIFIER
  - `IStatefulObject` persistence object identifier, value: "`uuid`"

Methods

- *addExtension*
  `public void` **addExtension**`( ICloneObjectExtension` **extension** `)`
  - **Usage**

    * Adds an `ICloneObjectExtension` to this clone object.
      The `ICloneObjectExtension` will be registered under the class returned by its `ICloneObjectExtension#getExtensionInterfaceClass()` method.

      There can only be one extension of a given type at any time. Adding an extension while another extension of the same type is already registered, will **replace** the existing extension.

      NOTE: in the current implementation stateful extensions will only be persisted for `IClone` objects.
  - **Parameters**
    * `extension` - the extension to add, will replace any existing extension of the same type, never null.
  - **See Also**
    * `ICloneObjectExtension`

- *clone*
  `public Object` **clone**`( )`
  - **Usage**
    * All implementations must be cloneable.

- *equals*
  `public boolean` **equals**`( Object` **obj** `)`
  - **Usage**
    * Equality based on uuid.

- *equalsAll*
  `public boolean` **equalsAll**`( ICloneObject` **otherCloneObject** `)`
  - **Usage**
    * Checks not only the uuid but ALL data fields for equality.
  - **Parameters**
    * `otherCloneObject` - clone object to compare to, may be null, may be same instance
  - **Returns** - true if all fields are equal, false otherwise or if otherCloneObject is null.

- *getExtension*
  `public ICloneObjectExtension` **getExtension**`( Class` **extensionClass** `)`
  - **Usage**
    * Retrieves an `ICloneObjectExtension` which has been added to this clone object. If no extension of this type has been added, null is returned.
      A deleted extension will not be returned.
  - **Parameters**
    * `extensionClass` - the extension type to retrieve, never null
  - **Returns** - the extension in question or NULL if no such extension has been added
  - **See Also**
    * `ICloneObjectExtension`

- *getExtensions*
  `public List` **getExtensions**`( )`
  - **Usage**
    * Retrieves a list of all currently added `ICloneObjectExtension` s for this clone object which have not been deleted.
  - **Returns** - list of non-deleted `ICloneObjectExtensions`, never null.
  - **See Also**

    * `ICloneObjectExtension`

- *getUuid*
  `public String` **getUuid**`( )`
  - **Usage**
    * Retrieves the `uuid` which uniquely identifies this object.

      This value has to be initialised during the construction of an object.
  - **Returns** - unique identifier of this object, never null.

- *hasExtensions*
  `public boolean` **hasExtensions**`( )`
  - **Usage**
    * Cached boolean value which indicates whether there is currently any `ICloneObjectExtension` added to this clone object.
      The return value will be false if there is no non-deleted extension present. Deleted extensions are not taken into account.
      This values is persisted as part of the `ICloneObject` .
  - **Returns** - true if there is at least one extension for this object.

- *hashCode*
  `public int` **hashCode**`( )`
  - **Usage**
    * HashCode based on uuid.

- *isMarked*
  `public boolean` **isMarked**`( )`
  - **Usage**
    * Checks whether this clone object has been marked.

      A typical use for marks is to pass info on some kind of selection of clone objects from a list between method calls without having to create an additional list or having to add an `ICloneObjectExtension` .

      Marks are not persisted.

      NOTE: Marks have no meaning outside of the module which set them. They exist only due to performance considerations. If you need to store some data for a clone object which will be handled by other modules, you should use `ICloneObject#getExtension(Class)` instead.
      Whenever a clone object leaves your module, you should consider all marks to be lost/corrupted.
  - **Returns** - true if this instance has been marked.

- *removeExtension*
  `public void` **removeExtension**`( Class` **extensionClass** `)`
  - **Usage**
    * Removes any `ICloneObjectExtension` of the given type from this `ICloneObject` .
      The extension is added to the internal deletion registry.
  - **Parameters**
    * `extensionClass` - the interface type for which a registered extension should be removed, never null.
  - **See Also**
    * `IStoreCloneObject.getDeletedExtensions()`
    * `IStoreCloneObject.purgeDeletedExtensions()`

- *removeExtension*
  `public void` **removeExtension**`( ICloneObjectExtension` **extension** `)`
  - **Usage**
    - Removes **any** `ICloneObjectExtension` which matches the `ICloneObjectExtension#getExtensionInterfaceClass()` value of the given extension from this `ICloneObject` .
      Convenience method.
      This is a short hand for `#removeExtension(Class)` .
  - **Parameters**
    - `extension` - the `ICloneObjectExtension` for which any extension of equal type should be removed, never null.
  - **See Also**
    - `ICloneObject.removeExtension(Class)` (in 2.1.5, page 19)

- *setMarked*
  `public void` **setMarked**`( boolean` **marked** `)`
  - **Usage**
    - Marks or unmarks a clone object.

      Marks are not persisted.

      Be sure to read the limitations for the usage of marks, see: `#isMarked()`
  - **Parameters**
    - `marked` - whether this instance should be marked or not.
  - **See Also**
    - `ICloneObject.isMarked()`

- *toString*
  `public String` **toString**`( )`
  - **Usage**
    - All implementations should provide a meaningful toString() method for debugging purposes.

2.1.6 Interface **ICloneObjectExtension**

Clone object extensions can be used by 3rd party modules to contribute their own data to any `ICloneObject` . Once added, such extensions will be persisted and synchronised by CPC automatically.

Any implementation needs to implement `ICloneObjectExtensionInterfaces` . Implementing only `ICloneObjectExtension` is not enough!

This interface lists all methods which are available to all CPC plugins and 3rd party contributions.

Additional methods are defined by more specific sub-interfaces which belong to individual CPC plugins and are to be considered private.
Any CPC plugin other than the one designated in the sub-interface API must not access such methods.

General Considerations valid for all `ICloneObjectExtension` implementations:
- All implementations must be **serializable** and **cloneable**.
- All implementations must be **registered with the `ICloneFactoryProvider`** 's corresponding extension point.
- An `ICloneObjectExtension` implementation is only valid for **one** `ICloneObject` type. The type needs to be specified during registration with the `ICloneFactoryProvider` . The same class may not be registered multiple times.

Declaration

public interface ICloneObjectExtension
implements ICloneDataElement, Cloneable, java.io.Serializable

Methods

- *clone*
  `public Object` **clone**`( )`
  - **Usage**
    - All implementations must be cloneable.

- *getExtensionInterfaceClass*
  `public Class` **getExtensionInterfaceClass**`( )`
  - **Usage**
    - Returns the `ICloneObjectExtension` sub-interface which this class is implementing.
      This value has to match the interface for which the implementation was registered with the `ICloneFactoryProvider` .

      I.e. if `CloneModificationHistoryExtensionImpl` implements `ICloneModificationHistoryExtension` , it would return `ICloneModificationHistoryExtension.class` here.

      This information could be obtained via reflection. However, as the interface class is potentially used as a `Map` key internally, the value needs to match exactly. Using reflection under these conditions would be error prone.
  - **Returns** - the interface class which this extension is implementing, never null.

- *isPartial*
  `public boolean` **isPartial**`( )`
  - **Usage**
    - Checks whether this `ICloneObjectExtension` object was fully restored from persistent storage.
      A true value indicates that there may be additional sub-element data available in persistent storage which was not loaded when this element was created.

      For all extensions which do not implement `ICloneObjectExtensionLazyMultiStatefulObject` this method always returns `false`.

      A newly created extensions (which are thus not yet persisted) which implement `ICloneObjectExtensionLazyMultiStatefulObject` should return `true` until `ICloneObjectExtensionLazyMultiStatefulObject#setPartial(boolean)` is used to set a new value.
  - **Returns** - true if this extension object may not have been fully restored, false otherwise.
  - **See Also**
    - `ICloneObjectExtensionLazyMultiStatefulObject.setPartial(boolean)` (in 5.1.1, page 32)

- *setParentUuid*
  `public void` **setParentUuid**`( String` **parentUuid** `)`
  - **Usage**

  - * This method will automatically be called, whenever an `ICloneObjectExtension` is added to an `ICloneObject` via `ICloneObject#addExtension(ICloneObjectExtension)` .
      A `ICloneObjectExtension` may only belong to one `ICloneObject` instance at a time and may not be reused.
      Should this method be called multiple times with different `parentUuids` an `IllegalArgumentException` is thrown.

      IMPORTANT: this method may only be called by the `ICloneObject` implementation.
  - **Parameters**
    * `parentUuid` - the UUID of the parent `ICloneObject` of this extension object, never null.
  - **Exceptions**
    * `IllegalArgumentException` - if a client ties to change the `parentUuid` of this object.

- *toString*
  `public String` **`toString`**`( )`
  - **Usage**
    * All implementations should provide a meaningful toString() method for debugging purposes.

2.1.7 Interface **ICloneObjectSupport**

This is a super interface for all additional support interfaces/classes which are not themselves `ICloneObject` implementations but which are non the less part of the CPC Clone Data objects.

Rationale:

> This interface exists only to provide type safety for calls where a distinction between `ICloneObject` s and other `ICloneDataElement` s is needed.

Declaration

public interface ICloneObjectSupport
implements ICloneDataElement, java.io.Serializable, Cloneable

2.2 Classes

2.2.1 Class **IClone.State**

Specifies the state of a clone.
The state influences the way a clone is handled by some CPC modules.
Additional information may be attached to a state.

Declaration

public static final class IClone.State
extends Enum

Fields

- public static final IClone.State DEFAULT
  - The clone was not modified or the modifications were judged not to be of consequence by the CPC Notification module.

    Furthermore the **same holds for all members of this clone's clone group**. Meaning that this clone and all it's clone group members are semantically equal or very nearly so.

    Another way of looking at this would be to consider `State#DEFAULT` as `IN SYNC` and all other states (besides `State#IGNORE`) as `NOT IN SYNC`.

    The CPC ruler will display these clones in green.
    No marker is added for clones of this this state.
    This is the initial state for all newly created clone instances.
- public static final IClone.State CUSTOMISED
  - The clone was not modified after the modifications made during its initial creation. Any changes made right after the clone was created are considered parametrisations of the clone and do not represent modifications which need to be propagated to other group members.

    The same state applies to all members of this clone's clone group.

    No marker is added for clones of this state.
- public static final IClone.State MODIFIED
  - The clone was modified and the modification was judged to be noteworthy by the CPC Notification module. However, it was not enough to warrant a state of `State#NOTIFY` or even `State#WARN` .

    Another use for this state are clones which had one of their clone group members modified. Once any member of a clone group changes to state `State#MODIFIED` , `State#NOTIFY` or `State#WARN` , all other members of the clone group which are currently in state `State#DEFAULT` should be changed into this state.

    The CPC ruler will display these clones in blue.
    No marker is added for clones of this this state.
- public static final IClone.State NOTIFY
  - A notification of type NOTIFY is pending for this clone.

    The CPC ruler will display these clones in yellow.
    An information marker is added for clones of this this state.
- public static final IClone.State WARN
  - A notification of type WARN is pending for this clone.

    The CPC ruler will display these clones in red.
    A warning marker is added for clones of this this state.
- public static final IClone.State IGNORE
  - The user requested that this clone should be ignored from now on.
    Its position will still be tracked and it will still be shown in clone views but at no point will any notifications or warnings be issued for a clone of this state.

    If this clone is member of a clone group, notifications will still be generated for changes within the other members of that group. However, the ignored clone will not be taken into account for the evaluation of modifications in other group members. In effect this is very similar to making the ignored clone leave the group. The only difference is that the clone could be "unignored" later and would rejoin its clone group as a normal group member.

The CPC ruler will display these clones in gray.

No marker is added for clones of this this state.

- public static final IClone.State ORPHAN
  - The clone is the only member of its group.

    Depending on the CPC configuration such clones will be periodically purged or are kept forever (research purposes). It is also up to the CPC configuration whether such "standalone" clones will be displayed to the user or whether they are hidden.

Chapter 3

Package core.api.data.collection

Package Contents *Page*

3.1 Interfaces

3.1.1 Interface **ICloneFileInterfaces**

Includes all interfaces which are required for an `ICloneFile` implementation.
Convenience collection interface.

Declaration

public interface ICloneFileInterfaces
implements core.api.data.ICloneFile, core.api.data.special.ICreatorCloneFile, core.api.data.special.IRemoteStoreCloneFile, ICloneObjectInterfaces

3.1.2 Interface **ICloneGroupInterfaces**

Includes all interfaces which are required for an `ICloneGroup` implementation.
Convenience collection interface.

Declaration

public interface ICloneGroupInterfaces
implements core.api.data.ICloneGroup, ICloneObjectInterfaces

3.1.3 Interface **ICloneInterfaces**

Includes all interfaces which are required for an `IClone` implementation.
Convenience collection interface.

Declaration

public interface ICloneInterfaces
implements core.api.data.IClone, core.api.data.special.ICreatorClone, ICloneObjectInterfaces

3.1.4 Interface **ICloneObjectExtensionInterfaces**

Includes all interfaces which are required for an `ICloneObjectExtension` implementation.
Convenience collection interface.

Declaration

public interface ICloneObjectExtensionInterfaces
implements core.api.data.ICloneObjectExtension

3.1.5 Interface **ICloneObjectInterfaces**

Includes all interfaces which are required for an `ICloneObject` implementation.
Convenience collection interface.

Declaration

public interface ICloneObjectInterfaces
implements core.api.data.ICloneObject, core.api.data.special.IStoreCloneObject

Chapter 4

Package core.api.data.extension

4.1 Interfaces

4.1.1 Interface **ICloneModificationHistoryExtension**

Extension object which is used by the `IStoreProvider` to attach `CloneDiff` data of all modifications since the last event on generation of `CloneModificationEvent` s and which can also be used to retrieve a list of all modifications made to a clone since its creation.

The `IStoreProvider` will also process this extension for all calls to `IStoreProvider#addClone(IClone)` or `IStoreProvider#updateClone(IClone, UpdateMode)` .
Any `CloneDiff` objects present will be added to the internal modification history of the store provider and can be retrieved by a call to `IStoreProvider#getFullCloneObjectExtension(ICloneObject, Class)` at any time.

This extension is persisted and lazily restored, see `ICloneObjectExtensionLazyMultiStatefulObject` .

Used by the `IStoreProvider` and the CPC Notification module.

Important: All implementations of this interface also need to implement `IStoreCloneModificationHistoryExtension` **!**

Declaration

public interface ICloneModificationHistoryExtension
implements core.api.data.ICloneObjectExtension

METHODS

- *addCloneDiff*
  `public void` **addCloneDiff**`( CloneDiff` **cloneDiff** `)`
  - **Usage**
    - Adds a new `CloneDiff` object to this clone modification history extension.
      `CloneDiff` s may only be added in correct time order (youngest first).
  - **Parameters**
    - `cloneDiff` - the clone diff to add, never null.
  - **Exceptions**
    - `IllegalArgumentException` - if `cloneDiff` is younger than the oldest existing `CloneDiff` element.

- *addCloneDiffs*
  `public void` **addCloneDiffs**`( List` **cloneDiffs** `)`
  - **Usage**
    - Adds a list of new `CloneDiff` objects to this clone modification history extension.
      `CloneDiff` s may only be added in correct time order (youngest first).
  - **Parameters**
    - `cloneDiffs` - a list of `CloneDiff` objects to add, never null.
  - **Exceptions**
    - `IllegalArgumentException` - if an added `cloneDiff` is younger than the oldest existing `CloneDiff` element.

- *addCloneDiffs*
  `public void` **addCloneDiffs**`( SortedSet` **cloneDiffs** `)`
  - **Usage**
    - Adds a list of new `CloneDiff` objects to this clone modification history extension.
      `CloneDiff` s may only be added in correct time order (youngest first).
  - **Parameters**
    - `cloneDiffs` - a sorted set of `CloneDiff` objects to add, never null.
  - **Exceptions**
    - `IllegalArgumentException` - if an added `cloneDiff` is younger than the oldest existing `CloneDiff` element.

- *clearCloneDiffs*
  `public void` **clearCloneDiffs**`( )`
  - **Usage**
    - Removes all `CloneDiff` objects from this clone modification history extension.
      This method may only be used if this extension was fully loaded
      (`ICloneObjectExtension#isPartial()` is false).

      This method is typically used when the clone history is either to be completely purged or to be replaced by some composite diffs.
  - **Exceptions**
    - `IllegalStateException` - is `ICloneObjectExtension#isPartial()` is true.

- *getCloneDiffs*
  `public List` **getCloneDiffs**`( )`
  - **Usage**
    - Yields a list of all modifications (`CloneDiff` s) made to this clone object.
      The returned list or its contents may not be modified in any way.

  - **Returns** - a list of clone diffs, may not be modified, may be empty, never null.

- *getCloneDiffsForTransaction*
  `public List` **getCloneDiffsForTransaction**`( )`
  - **Usage**
    - Yields a list of all modifications (`CloneDiff` s) made to this clone object since the last call of `IStoreCloneModificationHistoryExtension#endOfTransaction()` .
      This means the list contains all `CloneDiff` s which were added to this extension since the last `CloneModificationEvent` in which this clone was marked as modified.
      The returned list or its contents may not be modified in any way.
  - **Returns** - a list of clone diffs, may not be modified, may be empty, never null.

- *getValidCreationDate*
  `public Date` **getValidCreationDate**`( )`
  - **Usage**
    - Returns the the next valid creation date for use in a `CloneDiff` element for this extension.

      This is either the current time or, if another `CloneDiff` element with that creation date already exists, the current time incremented by a couple of milliseconds.

      This is important in order for a user of this extension to ensure that no two `CloneDiff` elements with the same creation date are added.
  - **Returns** - valid creation date for a `CloneDiff` element, never null.

4.1.2 Interface **ICloneNonWsPositionExtension**

A special non-whitespace position object for `IClone` objects.
Contains start and end offset with whitespaces (,“t,“n,“r) ignored.

This extension is not persisted.

Used by the `CoreClonePositionUtils` and the CPC Reconciler module.

Declaration

public interface ICloneNonWsPositionExtension
implements core.api.data.ICloneObjectExtension

Methods

- *getEndNonWsOffset*
  `public int` **getEndNonWsOffset**`( )`
  - **Usage**
    - Retrieves the non-whitespace end offset for this clone.

      IMPORTANT: This value is not persisted and not automatically updated during document modifications.
      If you need these values, you should update them yourself by calling:
      `CoreClonePositionUtils#extractPositions(core.api.provider.data.ICloneFactoryProvider, java.util.List, String)`

  – **Returns** - character offset to the beginning of the file, at which the clone ends (whitespaces not included!). The character at this position is still part of the clone. First char is 0.

- *getStartNonWsOffset*
  `public int` **getStartNonWsOffset**`( )`
  – **Usage**
    * Retrieves the non-whitespace start offset for the clone.

      IMPORTANT: This value is not persisted and not automatically updated during document modifications.
      If you need these values, you should update them yourself by calling:
      `CoreClonePositionUtils#extractPositions(ICloneFactoryProvider, List, String)`
  – **Returns** - the character offset to the beginning of the file at which the clone begins (whitespaces not included!). The character at this position is already part of the clone. First char is 0.

- *setEndNonWsOffset*
  `public void` **setEndNonWsOffset**`( int` **endNonWsOffset** `)`
  – **Usage**
    * Sets this extensions non-whitespace end offset.
  – **See Also**
    * `ICloneNonWsPositionExtension.getEndNonWsOffset()`

- *setStartNonWsOffset*
  `public void` **setStartNonWsOffset**`( int` **startNonWsOffset** `)`
  – **Usage**
    * Sets this extensions non-whitespace start offset.
  – **See Also**
    * `ICloneNonWsPositionExtension.getStartNonWsOffset()`

Chapter 5

Package core.api.data.special

Package Contents *Page*

5.1 Interfaces

5.1.1 Interface ICloneObjectExtensionLazyMultiStatefulObject

Extension of the `ICloneObjectExtensionMultiStatefulObject` interface.
By implementing this interface, an `ICloneObjectExtension` indicates that the additional sub-elements should not be automatically restored from the database during a normal lookup.

This is especially useful for `ICloneObjectExtension` s with a large number of sub-elements which would use up considerable amounts of memory, if they were always loaded into memory by default.
One example is the `ICloneModificationHistoryExtension` .

The `IStoreProvider#getFullCloneObjectExtension(core.api.data.ICloneObject, Class)` method can be used to retrieve all sub-element data for a lazy loaded extension object.

Declaration

public interface ICloneObjectExtensionLazyMultiStatefulObject
implements ICloneObjectExtensionMultiStatefulObject

Methods

- *setPartial*
  `public void` **setPartial**`( boolean` **partial** `)`
  - **Usage**
    - Sets the partial state of this extension.

      IMPORTANT: This method must only be used by the persistence provider.
  - **Parameters**
    - `partial` - true if there **might** be additional sub-element data in persistent storage, false otherwise.
  - **See Also**
    - `ICloneObjectExtensionLazyMultiStatefulObject.isPartial()`

5.1.2 Interface **ICloneObjectExtensionMultiStatefulObject**

Extension interface for `ICloneObjectExtensionStatefulObject` for `ICloneObjectExtension` implementations.
This interface is meant used for `ICloneObjectExtension` s which need to persist a large, fluctuating number of fields.
The `ICloneObjectExtensionStatefulObject` API requires you to map your internal state to a single `Map<String,String>` object with a predefined, unchanging number of values. This does offer you any good way of persisting lists with complex content of arbitrary length.

If you just want to persist a simple object with a fixed number of fields, please refer to the `ICloneObjectExtensionStatefulObject` API.

An object of this type is persisted/restored in two steps:
1. The regular part of the object is persisted/restored like normal `ICloneObjectExtensionStatefulObject` object.
2. The all list parts of the object are persisted/restored.

There can be an arbitrary but fixed number of different sub-element types. The methods of this interface always return lists or elements or of lists. Each entry in the top most list corresponds to one sub-element type.
The order of sub-element types must be the same for all methods.

I.e. if you have an extension X which keeps a couple of fields as well as two lists, one holding elements of type A the other of type B. To persist X you would implement `ICloneObjectExtensionMultiStatefulObject` and use the normal `ICloneObjectExtensionStatefulObject` API for the persistence of all the fields of X, except the lists.
All methods of this API would then return/take a two element top most list, the first element corresponding to persistence data for A and the second element corresponding to persistence data for B.

- `ICloneObjectExtensionMultiStatefulObject#getMultiPersistenceClassIdentifier()` would yield a list, i.e.: {"classid_a", "classid_b"}.

- `ICloneObjectExtensionMultiStatefulObject#getMultiPersistenceObjectIdentifier()` would yield a list, i.e.: {"creationDate", "somefield"}.
- `ICloneObjectExtensionMultiStatefulObject#getMultiState()` would yield a list of lists. The top most list contains two lists, the first contains states for all sub-elements of type A and the second list states for all sub-elements of type B.
- ...

If you only have just one sub-element type, just return a list with a single entry, as top most list.
If you don't have any sub-element, you should look at `ICloneObjectExtensionStatefulObject` instead.

Declaration

public interface ICloneObjectExtensionMultiStatefulObject
implements ICloneObjectExtensionStatefulObject

Fields

- public static final String DELETION_MARK_IDENTIFIER
  - The `ICloneObjectExtensionMultiStatefulObject#getMultiState()` key to use to indicate whether an `ICloneObjectExtensionMultiStatefulObject` sub-element was deleted.
    The type for this key is Boolean.
    The key is optional, if not present, the default value is `false`.

    IMPORTANT: this key must not be used for anything else. It must also not be part of the `ICloneObjectExtensionMultiStatefulObject#getMultiStateTypes()` maps.

Methods

- *getMultiPersistenceClassIdentifier*
  `public List` **getMultiPersistenceClassIdentifier**`( )`
  - **Usage**
    * A list of `PERSISTENCE_CLASS_IDENTIFIER` for the sub-objects.
  - **Returns** - a list of `PERSISTENCE_CLASS_IDENTIFIER` for the sub-objects, never null.
  - **See Also**
    * `IStatefulObject.getPersistenceClassIdentifier()`

- *getMultiPersistenceObjectIdentifier*
  `public List` **getMultiPersistenceObjectIdentifier**`( )`
  - **Usage**
    * A list of `PERSISTENCE_OBJECT_IDENTIFIER` for the sub-objects.
      The corresponding key in the state map has to be a unique value.
  - **Returns** - a list of `PERSISTENCE_OBJECT_IDENTIFIER` for the sub-objects, never null.
  - **See Also**
    * `IStatefulObject.getPersistenceObjectIdentifier()`

- *getMultiState*
  `public List` **getMultiState**`( )`
  - **Usage**

    - Retrieves a list of lists of `IStatefulObject#getState()` mappings for each sub-element of this extension.

      The elements of the topmost list must be **sorted** in ascending order according to the key returned by `ICloneObjectExtensionMultiStatefulObject#getMultiPersistenceObjectIdentifier()` .

      The "`parent_uuid`" key needs to be present in each mapping.

      The list also needs to contain deleted sub-elements. For those the `DELETION_MARK_IDENTIFIER` key in the state map should be set to `true`.
  - **Returns** - list of `IStatefulObject#getState()` mappings, never null.
  - **See Also**
    - `IStatefulObject.getState()`

- *getMultiStateTypes*
  `public List` **getMultiStateTypes**`( )`
  - **Usage**
    - Retrieves the `IStatefulObject#getStateTypes()` mapping for the different sub-element types.
      The "`parent_uuid`" key needs to be present in all mappings.
  - **Returns** - `IStatefulObject#getStateTypes()` mapping for sub-element type.
  - **See Also**
    - `IStatefulObject.getStateTypes()`

- *purgeDeletedEntries*
  `public void` **purgeDeletedEntries**`( )`
  - **Usage**
    - Indicates that all deleted sub-elements can be purged.
      After this method was called, `ICloneObjectExtensionMultiStatefulObject#getMultiState()` should contain only non-deleted sub-entries.

      This method is typically called by the store provider once this `ICloneObjectExtensionStatefulObject` was successfully persisted.

- *setMultiState*
  `public void` **setMultiState**`( List` **states** `)`
  - **Usage**
    - Sets the internal state of this object and all its sub-objects via the given list of lists of `IStatefulObject#setState(Map)` mappings.

      If the parameter is NULL, all sub-element data for this extension object is cleared.
      This typically happens when all clone data for a file is persisted.
  - **Parameters**
    - `states` - list of `IStatefulObject#setState(Map)` mappings, may be NULL.
  - **See Also**
    - `IStatefulObject.setState(Map)` (in 5.1.8, page 41)

5.1.3 INTERFACE ICloneObjectExtensionStatefulObject

A special version of the `IStatefulObject` interface which needs to be implemented by all `ICloneObjectExtension` objects which require persistence.

If you need to persist extension objects which contain lists of arbitrary length or complex content, please refer to the `ICloneObjectExtensionMultiStatefulObject` API.

There are a number of important differences between the normal `IStatefulObject` handling for `ICloneObject` s & co and the handling for `ICloneObjectExtension` s.

- The `IStatefulObject#getPersistenceObjectIdentifier()` **must** be "`parent_uuid`".
- The value for the key "`parent_uuid`" as returned by `IStatefulObject#getState()` **must** correspond to the `parent UUID` as set via `ICloneObjectExtension#setParentUuid(String)` .
- There can only be one `ICloneObjectExtensionStatefulObject` object per `ICloneObject` , the `parent UUID` is thus a unique identifier for any extension instance of a given type.
- An `ICloneObjectExtension` is limited to exactly one, pre-specified `ICloneObject` type. The `PERSISTENCE_CLASS_IDENTIFIER` of that type needs to be returned by `ICloneObjectExtensionStatefulObject#getPersistenceParentClassIdentifier()` method.

NOTE: In the current implementation stateful extensions are only meaningful for `IClone` objects.

Declaration

public interface ICloneObjectExtensionStatefulObject
implements IStatefulObject, core.api.data.ICloneObjectExtension

Fields

- public static final String PERSISTENCE_OBJECT_IDENTIFIER
  –

Methods

- *getPersistenceParentClassIdentifier*
  `public String` **getPersistenceParentClassIdentifier**`( )`
  - **Usage**
    * Each `ICloneObjectExtensionStatefulObject` implementation has to be linked to one specific `ICloneObject` type.
      This method must return the `PERSISTENCE_CLASS_IDENTIFIER` value of that class.
      I.e. `IClone#PERSISTENCE_CLASS_IDENTIFIER` .
  - **Returns** - the `IStatefulObject#getPersistenceClassIdentifier()` of the parent entity type, never null.

- *isDirty*
  `public boolean` **isDirty**`( )`
  - **Usage**
    * Checks whether this `ICloneObjectExtensionStatefulObject` instance was modified in a way which affected the persistent part of its data.
      This should be true if and only if a modification has taken place which might have resulted in a change of the `IStatefulObject#getState()` return value.

      For `ICloneObjectExtensionMultiStatefulObject` implementations, true should also be returned if a modification might have changed the return value of `ICloneObjectExtensionMultiStatefulObject#getMultiState()` .
  - **Returns** - true if this instance was modified in any way which needs to be persisted, false otherwise.

- *setDirty*
  `public void` **setDirty**`( boolean` **dirty** `)`
  - **Usage**
    - Called by the `IStoreProvider` (with value `false`) after this extension was successfully persisted.

      IMPORTANT: this method may only be called internally or by the `IStoreProvider` .
  - **Parameters**
    - `dirty` - true if this entry is out of sync and needs to be persisted, false otherwise.

5.1.4 Interface **ICreatorClone**

Internal sub-interface of `IClone` containing internal methods which are related to the creation of new clone objects as well as the modification of a clone contents.

This interface may only be used by modules which create or modify clone objects.

Users:
- CPC Track
- CPC Imports

Declaration

public interface ICreatorClone
implements core.api.data.IClone

Methods

- *setContent*
  `public void` **setContent**`( String` **content** `)`
  - **Usage**
    - Sets the current content for the position range specified by the clone.
      This method is called on creation of a new clone and after each modification to the content of a clone.
      The first call of this method will also set the `IClone#getOriginalContent()` value.
      This method also sets the `IClone#getModificationDate()` value to the current time.
  - **Parameters**
    - `content` - the new clone content, never null.
  - **See Also**
    - `IClone.getModificationDate()`

- *setCreationDate*
  `public void` **setCreationDate**`( Date` **creationDate** `)`
  - **Usage**
    - Sets the creation date of this clone.
      This method will also set the `IClone#getModificationDate()` and
      `IClone#getCloneStateChangeDate()` values to the given date.
  - **Parameters**
    - `creationDate` - the creation date, never null.
  - **See Also**
    - `IClone.getModificationDate()`

    * `IClone.getCloneStateChangeDate()`

- *setCreator*
  `public void` **setCreator**`( String` **creator** `)`
  - **Usage**
    * Sets the creator (username) of this clone.
      If the creator can't be determined, the value should be set to NULL.
  - **Parameters**
    * `creator` - the creator of this clone, may be NULL.

- *setFileUuid*
  `public void` **setFileUuid**`( String` **fileUuid** `)`
  - **Usage**
    * Sets the UUID for the clone file in which this clone is located.
  - **Parameters**
    * `fileUuid` - clone file uuid, never null.

5.1.5 Interface **ICreatorCloneFile**

Internal sub-interface of `ICloneFile` containing internal methods which are related to the creation of new clone file objects as well as any kind of modification.

This interface may only be used by modules which create or modify clone file objects.

Users:
- CPC Store
- CPC Store Remote?

Declaration

public interface ICreatorCloneFile
implements core.api.data.ICloneFile

Methods

- *setModificationDate*
  `public void` **setModificationDate**`( long` **modificationDate** `)`
  - **Usage**
    * Set by `IStoreProvider` during a call to `IStoreProvider#persistData(ICloneFile)` .
  - **Parameters**
    * `modificationDate` - the last modification date of the file.

- *setPath*
  `public void` **setPath**`( String` **path** `)`
  - **Usage**
    * Set by `IStoreProvider` at `ICloneFile` creation time and on file moves.
  - **Parameters**
    * `path` - the project relative path of the file, never null.

- *setProject*
  `public void` **setProject**`( String` **project** `)`
  - **Usage**
    - Set by `IStoreProvider` at `ICloneFile` creation time and on file moves.
  - **Parameters**
    - `project` - the project name, never null.

- *setSize*
  `public void` **setSize**`( long` **size** `)`
  - **Usage**
    - Set by `IStoreProvider` during a call to `IStoreProvider#persistData(ICloneFile)` .
  - **Parameters**
    - `size` - the file size in bytes.

5.1.6 Interface **IMultiKeyStatefulObject**

This interface is for internal use only.
It is not intended to be implemented by any client of the CPC framework.

Declaration

public interface IMultiKeyStatefulObject
implements IStatefulObject

Methods

- *getPersistenceObjectIdentifiers*
  `public List` **getPersistenceObjectIdentifiers**`( )`
  - **Usage**
    - Retrieves a list of keys which together form a unique identifier for the object.
  - **Returns** - list of keys which form a unique identifier, never null.
  - **See Also**
    - `IStatefulObject.getPersistenceObjectIdentifier()`

5.1.7 Interface **IRemoteStoreCloneFile**

Extension interface for `ICloneFile` which provides access to internal data fields for use only by an `IStoreProvider` .

All `ICloneFile` implementations have to implement this interface.

Rationale:

> These fields should not be accessed by other plugins. They are therefore "hidden" by this extra interface. The fact that an `ICloneFile` object will need to be cast to this interface before any of the fields can be accessed is meant to work as a deterrent for accidental access to these fields.
>
> The `ICloneObjectExtension` mechanism is not used by the base CPC plugins for performance reasons.

Declaration

```
public interface IRemoteStoreCloneFile
implements core.api.data.ICloneFile
```

Methods

- *getRepositoryVersion*
  `public String` **getRepositoryVersion**`( )`
  - **Usage**
    - The current repository version number for the file underlying this `ICloneFile` object. This value equals the ones in `EclipseTeamEvent#getNewRevision()` .
  - **Returns** - repository version or NULL if not set

- *isRemoteDirty*
  `public boolean` **isRemoteDirty**`( )`
  - **Usage**
    - Whether the clone data for this file was locally modified since the last sync with the repository and must therefore be sent to the repository with the next commit.

- *setRemoteDirty*
  `public void` **setRemoteDirty**`( boolean` **remoteDirty** `)`
  - **Usage**
    - Sets the remote dirty flag for this clone file.
  - **See Also**
    - `IRemoteStoreCloneFile.isRemoteDirty()`

- *setRepositoryVersion*
  `public void` **setRepositoryVersion**`( String` **repositoryVersion** `)`
  - **Usage**
    - This value equals the ones in `EclipseTeamEvent#getNewRevision()` .
  - **Parameters**
    - `repositoryVersion` - the repository version to set, may be null

5.1.8 Interface **IStatefulObject**

Special sub-interface for all objects which are to be persisted by an `IStoreProvider` . The methods are used to extract and restore the internal state of an object.

Declaration

```
public interface IStatefulObject
```

Methods

- *getPersistenceClassIdentifier*
  `public String` **`getPersistenceClassIdentifier( )`**
  - **Usage**
    - Returns a string which uniquely identifies the object type.

      For the default `ICloneObject` sub interfaces the return values **must equal** the `PERSISTENCE_CLASS_IDENTIFIER` constant defined in the interface.

      3rd party clone objects can define their own persistence identifiers.
      Allowed are only letters, numbers and the underscore. Furthermore an identifier needs to contain at least one letter, may not begin with an underscore and may not contain multiple consecutive underscores.

      A typical store provider will use this method to derive directory/file or table names.
  - **Returns -** unique identifier for the object type, never null.

- *getPersistenceObjectIdentifier*
  `public String` **`getPersistenceObjectIdentifier( )`**
  - **Usage**
    - Returns a key which corresponds to an entry in the state Map returned by `getState()` which uniquely identifies an object instance.

      For the default `ICloneObject` sub interfaces the return values **must equal** the `PERSISTENCE_OBJECT_IDENTIFIER` constant defined in the `ICloneObject` interface.

      By default this is `uuid`.

      3rd party clone objects can define their own persistence identifiers.
      Allowed are only letters, numbers and the underscore. Furthermore an identifier needs to contain at least one letter, may not begin with an underscore and may not contain multiple consecutive underscores.

      A typical store provider will use this method to derive file names, internal id structure names or table primary keys.

      IMPORTANT: the corresponding value in `IStatefulObject#getState()` **must not** be changed at any point.
      The unique identifiers of all stateful objects must remain unchanged during their entire lifetime.
  - **Returns -** key for state Map which yields a unique identifier for an object instance, never null. Will usually return ”uuid”.
  - **See Also**
    - `IStatefulObject.getState()`

- *getState*
  `public Map` **`getState( )`**
  - **Usage**
    - Returns a map which fully describes the internal state of this object. The map is persisted by the `IStoreProvider` and the `setState()` method is used to restore a persisted state.

      In case of an `ICloneObject` , the map **does not** include any data for `ICloneObjectExtension` which are stored under the object.

      The values in the map are restricted to the following object types:

- String
- Integer
- Long
- Boolean
- java.util.Date
- null

The keys in the map should correspond to the following schema:

- Key names may not contain any characters other than letters, numbers, underscores and dots. Furthermore an identifier needs to contain at least one letter, may not begin with an underscore and may not contain multiple consecutive underscores.
- For simple fields the name of the field **has to** equal the name of the key in the map. I.e. getUuid()/setUuid(String) must use the key "uuid".
- For complex fields the name of the field should be used as a prefix, separated by a dot and then followed by any field names inside the complex field. I.e. getPosition().getStartOffset() should use the key "position.startOffset".

IMPORTANT NOTEs:

- The key set must be static. An implementation may not change the number or names of keys dynamically. I.e. lists may not be represented as a variable number of keys. They have to be encoded as a single String element. However, keys which would contain a null value may be left out.
- It is crucial that all implementations adhere to the naming scheme mentioned above as some key semantics are hard coded. (i.e. "uuid" is the primary key)
- The `IStatefulObject#getPersistenceObjectIdentifier()` entry in this state map must always be defined and its value **may not change** during the lifetime of a stateful object.

– **Returns** - a map describing the internal state of this object, never null.

– **See Also**

  ∗ `IStatefulObject.setState(Map)` (in 5.1.8, page 41)

- *getStateTypes*
  `public Map` **getStateTypes**`( )`

  – **Usage**

    ∗ Returns a list of **all** keys which are needed to persist this objects internal state. The values in the returned Map correspond to the classes of the values which will be used in the Map returned by `getState()`.

      I.e. if `getState()` will return a Map which contains the key "uuid" with the value "jda83ds-..." then this Map contains the key "uuid" with the value `String.class`.

  – **Returns** - map which contains **all** keys and their data types, never null

  – **See Also**

    ∗ `IStatefulObject.getState()`

- *setState*
  `public void` **setState**`( Map` **state** `)`

  – **Usage**

    ∗ Restores the internal state of this object to the state which was extracted by means of `getState()` earlier.

      For a description of the internal structure of the map, see `getState()`.

  – **Parameters**

    ∗ `state` - a map describing the internal state of this object, never null

  – **See Also**

    ∗ `IStatefulObject.getState()`

5.1.9 INTERFACE IStoreCloneModificationHistoryExtension

Special extension interface for `ICloneModificationHistoryExtension` which contains methods which may only be called by the current `IStoreProvider` .

Declaration

public interface IStoreCloneModificationHistoryExtension
implements core.api.data.extension.ICloneModificationHistoryExtension

Methods

- *endOfTransaction*
  `public void` **endOfTransaction**`( )`
  - **Usage**
    - Indicates the end of the current `IStoreProvider` transaction.

      IMPORTANT: This method must only be used from within the `IStoreProvider` !
  - **See Also**
    - `ICloneModificationHistoryExtension.getCloneDiffsForTransaction()`
    - `IStoreProvider`

- *getEndOfTransactionCloneDiffCreationDate*
  `public Date` **getEndOfTransactionCloneDiffCreationDate**`( )`
  - **Usage**
    - Retrieves the creation date of the last (oldest) `CloneDiff` of this extension at the time of the last call of `#endOfTransaction()` .
      If `#endOfTransaction()` was never called or if no `CloneDiff` elements were added at that point in time, this method will return NULL.

      IMPORTANT: This method must only be used from within the `IStoreProvider` !
  - **Returns** - creation date of oldest diff during last call of `#endOfTransaction()` , may be NULL.

- *setEndOfTransactionCloneDiffCreationDate*
  `public void` **setEndOfTransactionCloneDiffCreationDate**`( Date` **endOfTransactionCloneDiffCreationDate** `)`
  - **Usage**
    - Sets the `#getEndOfTransactionCloneDiffCreationDate()` value.
      This method is used by the `IStoreProvider` to "merge" multiple `ICloneModificationHistoryExtension` s, if needed.

      IMPORTANT: This method must only be used from within the `IStoreProvider` !
  - **Parameters**
    - `endOfTransactionCloneDiffCreationDate` - the new end of transaction creation date, may be NULL.

- *wasCleared*
  `public boolean` **wasCleared**`( )`
  - **Usage**
    - Checks whether this extension was cleared (`ICloneModificationHistoryExtension#clearCloneDiffs()`) since the end of the last transaction.
      This value is set to `true` when `ICloneModificationHistoryExtension#clearCloneDiffs()` is called and is reset to `false` when `#endOfTransaction()` is called.
      The default value is `false`.
  - **Returns** - `true` if this extension was cleared recently, `false` otherwise.

5.1.10 Interface **IStoreCloneObject**

Extension interface for `ICloneObject` which contains additional internal methods for use only by an `IStoreProvider` .

All `ICloneObject` implementations have to implement this interface.

Rationale:

> These methods should not be accessed by other plugins besides the `IStoreProvider` . They are therefore "hidden" by this extra interface. The fact that an `ICloneObject` object will need to be cast to this interface before any of the methods can be accessed is meant to work as a deterrent for accidental access to these methods.
> The `ICloneObjectExtension` mechanism is not used by most CPC plugins for performance reasons.

Declaration

public interface IStoreCloneObject
implements core.api.data.ICloneObject, IStatefulObject

Methods

- *getDeletedExtensions*
  `public List` **getDeletedExtensions**`( )`
  - **Usage**
    - Retrieves a list of deleted `ICloneObjectExtension` s for this clone object.
  - **Returns** - list of deleted `ICloneObjectExtension`s, never null.
  - **See Also**
    - `IStoreCloneObject.purgeDeletedExtensions()`
    - `ICloneObject.removeExtension(Class)` (in 2.1.5, page 19)
    - `ICloneObject.removeExtension(ICloneObjectExtension)` (in 2.1.5, page 20)

- *isDirty*
  `public boolean` **isDirty**`( )`
  - **Usage**
    - Whether this clone was modified and will need to be written to persistent storage.
  - **Returns** - true if this instance was modified

- *isPersisted*
  `public boolean` **isPersisted**`( )`
  - **Usage**
    - Whether this clone was already stored in persistent storage at some point.
      This does not mean that it can't be dirty dirty.
      This is interesting for storage provider implementations which require different actions for addition and update of data (i.e. SQL INSERT and UPDATE).
  - **Returns** - true if this instance was persisted in the past

- *purgeDeletedExtensions*
  `public void` **purgeDeletedExtensions**`( )`
  - **Usage**

    * Purges all currently deleted extensions from the `ICloneObject` .
      After this method was called, the return value of `#getDeletedExtensions()` will be an empty list.

      A store provider will typically call this method each time an `ICloneObject` was persisted successfully (and there is thus no need to retain any information about deleted extensions any longer).
  - **See Also**
    * `IStoreCloneObject.getDeletedExtensions()`

- *setDirty*
  `public void` **setDirty**`( boolean` **dirty** `)`
  - **Usage**
    * Sets this clone objects dirty flag.
  - **See Also**
    * `IStoreCloneObject.isDirty()`

- *setPersisted*
  `public void` **setPersisted**`( boolean` **persisted** `)`
  - **Usage**
    * Sets this clone objects persisted flag.
  - **See Also**
    * `IStoreCloneObject.isPersisted()`

Chapter 6

Package core.api.hub.event

6.1 Classes

6.1.1 Class **CloneEvent**

Abstract base event for all clone data related events.

Declaration

```
public abstract class CloneEvent
extends core.api.hub.event.CPCEvent
```

Constructors

- *CloneEvent*
  `public` **CloneEvent**`( ICloneFile` **cloneFile** `)`
  - **Usage**
    - Create a new clone event, this abstract constructor needs to be called by all sub-implementations. The clone file value may be NULL if the event is not related to a specific file.
  - **Parameters**
    - `cloneFile` - the clone file which this event is related to, may be NULL.

Methods

- *getCloneFile*
  `public ICloneFile` **getCloneFile**`( )`
  - **Usage**
    - Retrieves the clone file for this event.
      If the event is not specifically related to a single clone file, this value is NULL.

      NOTE: In case of a file deletion this clone file entry may point to a no longer existing file and it may also be no longer possible to retrieve this file or any data about it from the store provider.
  - **Returns** - the clone file which this event is related to, may be NULL.

- *subToString*
  `protected String` **subToString**`( )`
  - **Usage**
    - Should be called as part of `Object#toString()` implementations of sub-classes.
  - **Returns** - data values from this class, never null.

6.1.2 Class **CloneModificationEvent**

Modification event object for clone data modifications. Send whenever the clone data for a file is changed for some reason.

NOTE:

- All values MAY BE NULL
- The same clone may be contained in more than one of the lists.
- This event represents a collection of all events which happened during one IStoreProvider "transaction". As such a clone may appear multiple times in different stages. I.e. it was added, moved and modified within a single transaction. If a clone was updated multiple times, it will be listed only once in the modified and moved clones lists (the latest version).
- A clone that is part of the deleted list, will not appear in any other list.
- If all clone data for a large number of files (or all clone data) is removed/updated a special format may be used. **`#getCloneFile()`** is NULL, **`#isFullModification()`** is true and all clone lists are NULL.
- All objects contained in this event are cloned. Thus instances of the same clone will not match between different lists. Use equals() to compare clone objects, not ==.
- **The clone objects must not be modified by a receiver.** They are shared between all receivers of this event.

IMPORTANT: Make sure you understand the effects of the `IStoreProvider.LockMode` value during the acquisition of an exclusive `IStoreProvider` lock before you create any events of this type yourself.

Declaration

public class CloneModificationEvent
extends core.api.hub.event.CloneEvent

Constructors

- *CloneModificationEvent*
  `public` **CloneModificationEvent**`( ICloneFile` **cloneFile** `)`
  - **Usage**
    - Creates a new `CloneModificationEvent` instance for dispatching via the event hub registry.
      i.e. {@code CPCCorePlugin.getEventHubRegistry().dispatch(newEvent);}

      The `cloneFile` parameter may be NULL, in the special case that this event is meant to indicate to all interested parties that **all** clone data has been potentially removed/updated.
      In this case fullModification has to be true and all lists must be null.
  - **Parameters**
    - `cloneFile` - the file for this event, if the event is specific to one file, may be NULL.
  - **See Also**
    - `IEventHubRegistry.dispatch(CPCEvent)` (in 7.1.2, page 72)
    - `CPCCorePlugin.getEventHubRegistry()`

Methods

- *getAddedClones*
  `public List` **getAddedClones**`( )`
  - **Usage**
    - A list of clones which were added during this event.
      They may also be moved and/or modified by the same event!
  - **Returns** - may be NULL

- *getModifiedClones*
  `public List` **getModifiedClones**`( )`
  - **Usage**
    - A list of clones which had their contents changed during this event.
      They may also be added and/or moved by the same event!
      A clone will not appear more than once within this list. It will only contain the latest version.

      Modified clones usually carry an `ICloneModificationHistoryExtension` object which contains `CloneDiff` data for all modifications since the last time the clone was part of an `CloneModificationEvent` .
      However, there are some special circumstances under which no modification history data is available for a clone which might have been modified. I.e. if a file is reverted or if a file was externally modified and it is not possible to generate an exact `CloneDiff` description of the change.
  - **Returns** - may be NULL
  - **See Also**
    - `ICloneModificationHistoryExtension`
    - `CloneDiff`

- *getMovedClones*
  `public List` **getMovedClones**`( )`
  - **Usage**
    - A list of clones which were moved during this event.
      They may also be added and/or modified by the same event!
      A clone will not appear more than once within this list. It will only contain the latest version.
  - **Returns** - may be NULL

- *getRemovedClones*
  `public List` **getRemovedClones**`( )`
  - **Usage**
    - A list of clones which were removed during this event.
      Any clone which is part of this list will not appear in any other list.
      A clone which was added and removed during a single transaction will not be part of any list.
  - **Returns** - may be NULL

- *isFullModification*
  `public boolean` **isFullModification**`( )`
  - **Usage**
    - Indicates whether the entire clone data for the file was modified.
      In this case all clone lists are NULL and a receiver should retrieve the latest clone data from the store provider.

      This typically happens when:
      - a file is reverted to an earlier state
      - ...
  - **Returns** - true if this event represents a full modification of the clone data

- *isValid*
  `public boolean` **isValid**`( )`

- *setAddedClones*
  `public void` **setAddedClones**`( List` **addedClones** `)`
  - **Usage**

    - Must only be called ONCE.
  - **Parameters**
    - `addedClones` - corresponding clone list, never null
  - **See Also**
    - `CloneModificationEvent.getAddedClones()`

- *setFullModification*
  `public void` **setFullModification**`( boolean` **fullModification** `)`
  - **Usage**
    - This may only be set to true if all clone lists of this event are null.
  - **Parameters**
    - `fullModification` - true if this event represents a full modification of the clone data
  - **See Also**
    - `CloneModificationEvent.isFullModification()`

- *setModifiedClones*
  `public void` **setModifiedClones**`( List` **modifiedClones** `)`
  - **Usage**
    - Must only be called ONCE.
  - **Parameters**
    - `modifiedClones` - corresponding clone list, never null
  - **See Also**
    - `CloneModificationEvent.getModifiedClones()`

- *setMovedClones*
  `public void` **setMovedClones**`( List` **movedClones** `)`
  - **Usage**
    - Must only be called ONCE.
  - **Parameters**
    - `movedClones` - corresponding clone list, never null
  - **See Also**
    - `CloneModificationEvent.getMovedClones()`

- *setRemovedClones*
  `public void` **setRemovedClones**`( List` **removedClones** `)`
  - **Usage**
    - Must only be called ONCE.
  - **Parameters**
    - `removedClones` - corresponding clone list, never null
  - **See Also**
    - `CloneModificationEvent.getRemovedClones()`

- *toString*
  `public String` **toString**`( )`

6.1.3 Class **CloneNotificationEvent**

Notification event object for clone modification warnings. Send whenever some module detected that a recent clone content modification might have introduced update anomalies.

This event type is typically generated by the `CPC Notification` module.
However, other modules are allowed to generate events of this type too.

The `CPC Notification UI` module will listen to events of this type and will display them to the user in some appropriate way.
Other modules may also listen to this event and may initiate their own actions.

Declaration

public class CloneNotificationEvent
extends core.api.hub.event.CloneEvent

Constructors

- *CloneNotificationEvent*
  `public` **CloneNotificationEvent**`( ICloneFile` **cloneFile** `)`
  - **Usage**
    - Creates a new `CloneNotificationEvent` for the given file.
  - **Parameters**
    - `cloneFile` - the file which contains the affected clone, never null.

Methods

- *getMessage*
  `public String` **getMessage**`( )`
  - **Usage**
    - Retrieves the message for this notification.
      This value is usually directly related to the message in the `IEvaluationResult` which triggered this event.
  - **Returns** - the message for this notification, may be NULL.
  - **See Also**
    - `IEvaluationResult.getMessage()`

- *getModifiedClone*
  `public IClone` **getModifiedClone**`( )`
  - **Usage**
    - Retrieves the clone instance which triggered this event.
  - **Returns** - the modified clone, never null.

- *getType*
  `public CloneNotificationEvent.Type` **getType**`( )`
  - **Usage**
    - Retrieves the type of this notification event.
      Specifies how this event should be presented to the user.

  - **Returns** - `Type` of this event, never null.

- *getWeight*
  `public double` **getWeight**`( )`
  - **Usage**
    - Retrieves the weight of this notification.
      This value is usually directly related to the weight in the `IEvaluationResult` which triggered this event.
  - **Returns** - the weight of this notification, always >=0.
  - **See Also**
    - `IEvaluationResult.getWeight()`

- *isValid*
  `public boolean` **isValid**`( )`

- *setMessage*
  `public void` **setMessage**`( String` **message** `)`
  - **Usage**
    - Retrieves the message for this notification.
      This value is usually directly related to the message in the `IEvaluationResult` which triggered this event.
  - **Parameters**
    - `message` - the message for this notification, may be NULL.

- *setModifiedClone*
  `public void` **setModifiedClone**`( IClone` **modifiedClone** `)`
  - **Usage**
    - Sets the clone instance which triggered this event.

      This is a required value.
  - **Parameters**
    - `modifiedClone` - the modified clone, never null.

- *setType*
  `public void` **setType**`( CloneNotificationEvent.Type` **type** `)`
  - **Usage**
    - Sets the type of this notification event.
      Specifies how this event should be presented to the user.

      This is a required value.
  - **Parameters**
    - `type` - the `Type` of this event, never null.

- *setWeight*
  `public void` **setWeight**`( double` **weight** `)`
  - **Usage**
    - Sets the weight of this notification.
      This value is usually directly related to the weight in the `IEvaluationResult` which triggered this event.
  - **Parameters**
    - `weight` - the weight of this notification, always >=0.
  - **See Also**
    - `IEvaluationResult.getWeight()`

- *toString*
  `public String` **toString**`( )`

6.1.4 Class **CloneNotificationEvent.Type**

Possible presentation styles for this notification event.
Closely related to `IEvaluationResult.Action` .

Declaration

public static final class CloneNotificationEvent.Type
extends Enum

Fields

- public static final CloneNotificationEvent.Type NOTIFY
  –
- public static final CloneNotificationEvent.Type WARN
  –
- public static final CloneNotificationEvent.Type DELAY_NOTIFY
  - Similar to `Type#NOTIFY` .
    However, this type indicates that the user should not be notified instantly but that the notification should be queued and that the modification should be reevaluated once the clone modification has finished.
    There are multiple potential approaches to this problem.
    I.e. notifications could be delayed until the user closes the corresponding editor or until the clone or the file it is located in has not been modified for a specific amount of time.
- public static final CloneNotificationEvent.Type DELAY_WARN
  - Similar to `Type#WARN` .

6.1.5 Class **ClonePersistenceEvent**

This event is generated by the `IStoreProvider` whenever the clone data is persisted to stable storage.
Typically this happens whenever the user saves a file. This event is also generated, if the clone data for a file is purged.

The `CloneEvent#getCloneFile()` value will be NULL if all clone data was purged.

IMPORTANT: This event is generated from within an exclusive write lock block inside of the `IStoreProvider` . A receiver is **not allowed** to make any calls to the store provider for the duration of the event dispatching.
Furthermore, as an exclusive lock is held, all receivers are urged to return as fast as possible.
Care should be taken to ensure that a receiver does not inadvertently trigger events which might lead to not absolutely necessary work being done during the lifetime of this event *(and therefore the exclusive write lock).*

Declaration

public class ClonePersistenceEvent
extends core.api.hub.event.CloneEvent

Constructors

- *ClonePersistenceEvent*
  `public` **ClonePersistenceEvent**`( ICloneFile` **cloneFile** `)`
  - **Usage**
    - Creates a new `ClonePersistenceEvent` for the given file.
  - **Parameters**
    - `cloneFile` - the file for which clone data was persisted, never null.

Methods

- *getClones*
  `public List` **getClones**`( )`
  - **Usage**
    - Retrieves a list with the new persisted clone data for this file.
      The list may be empty, if no clones are persisted for this file.

      IMPORTANT: the `IClone` instances may **not** me modified in any way.
  - **Returns** - a list with the latest versions of all `IClone` instances for this file, never null.

- *setClones*
  `public void` **setClones**`( List` **clones** `)`
  - **Usage**
    - Sets the list of persisted clones.

      NOTE: For performance reasons `IClone` instances do not need to be cloned or sealed for use in this event. Receivers of this event are not allowed to modify the instances in any way.
  - **Parameters**
    - `clones` - a list with the latest versions of all `IClone` instances for this file, never null.

- *toString*
  `public String` **toString**`( )`

6.1.6 Class **CPCEvent**

Abstract parent class for all types of CPC Events. This is the root of the event class hierarchy.

Comparability is defined based on the creation time of an event object.
Creation times are guaranteed to be unique.

Declaration

public abstract class CPCEvent
extends Object
implements Comparable, Cloneable

Constructors

- *CPCEvent*
  `public` **CPCEvent**`( )`
  - **Usage**
    - Creates a new `CPCEvent` instance with a unique creation time.

Methods

- *checkSeal*
  `protected void` **checkSeal**`( )`
  - **Usage**
    - Ensures that this event has not yet been sealed.
  - **Exceptions**
    - `IllegalStateException` - if the event was already sealed.

- *clone*
  `protected Object` **clone**`( )`

- *compareTo*
  `public int` **compareTo**`( CPCEvent` **o** `)`

- *getCreationTime*
  `public long` **getCreationTime**`( )`
  - **Usage**
    - Retrieves the creation time of this event in milliseconds.
      The value corresponds to `System#currentTimeMillis()` at the time of the creation of the event object.

      Creation times are guaranteed to be unique. Equality and comparability are based on event creation times.
  - **Returns** - creation time of this event.

- *isValid*
  `public boolean` **isValid**`( )`
  - **Usage**
    - Checks if this event has been fully initialised.
      Will return `false` if one of the mandatory fields of the event has not yet been filled out.

      Subclasses should override this method but should never return `true`. Instead they should delegate to the super class implementation once all validity checks on their level have passed.

      The `CPCEvent#isValid()` implementation always returns `true`.
  - **Returns** - `true` if this event is valid, `false` otherwise.
  - **See Also**
    - `IEventHubRegistry.dispatch(CPCEvent)` (in 7.1.2, page 72)

- *seal*
  `public void` **seal**`( )`
  - **Usage**

    * Marks this event as sealed.
      Once sealed, no more modifications to the contents of the event are allowed.
      This method may only be called once per event.

      IMPORTANT: An event is always sealed by the `IEventHubRegistry` once the event is being dispatched. The creator of the event, **must not** call this method.

      Trying to modify a sealed event will throw an IllegalStateException.

      Subclasses may override this method but must ensure that they call it in their new `seal()` implementation.

- *toString*
  `public abstract String` **toString**`( )`
  - **Usage**
    * Every event should implement a sensible toString method for use in debugging log messages.
  - **Returns** - debug string representation, never null.

6.1.7 Class EclipseCutCopyPasteEvent

This event is generated by the `CPC Sensor` module, whenever the programmer executes a cut, copy or paste operation.
Besides the type and position, the event provides information about the current selection, the clipboard and the content of the editor.

Declaration

public class EclipseCutCopyPasteEvent
extends core.api.hub.event.EclipseEvent

Constructors

- *EclipseCutCopyPasteEvent*
  `public` **EclipseCutCopyPasteEvent**`( String` **user**`, String` **project** `)`
  - **Usage**
    * Creates a new `EclipseCutCopyPasteEvent` for the given user and project.
  - **Parameters**
    * `user` - the current user, never null.
    * `project` - the project for the file affected by this operation, never null.

Methods

- *getClipboard*
  `public String` **getClipboard**`( )`
  - **Usage**
    * Retrieves the current clipboard content.
  - **Returns** - current clipboard content, never null.

- *getEditorContent*
  `public String` **getEditorContent**`( )`

  - **Usage**
    - Retrieves the current content of the file/editor which was affected by this operation.
  - **Returns** - current editor content, never null.

- *getOffset*
  `public int` **getOffset**`( )`
  - **Usage**
    - Retrieves the offset in the document at which the operation occurred.
      If the current selection is not empty, this is also the start offset of the selection. For a paste operation, the clipboard is inserted at this position.
      The offset is the zero-based position character position within the file.
  - **Returns** - offset of this event, always >=0.

- *getSelection*
  `public String` **getSelection**`( )`
  - **Usage**
    - Retrieves the current selection in the editor.
  - **Returns** - the current selection in the editor, never null.

- *getType*
  `public EclipseCutCopyPasteEvent.Type` **getType**`( )`
  - **Usage**
    - Retrieves the `EclipseCutCopyPasteEvent.Type` of this event.
  - **Returns** - type of this event, never null.

- *isValid*
  `public boolean` **isValid**`( )`

- *setClipboard*
  `public void` **setClipboard**`( String` **clipboard** `)`
  - **Usage**
    - Sets the current clipboard content.

      This is a required value.
  - **Parameters**
    - `clipboard` - the current clipboard content, never null.

- *setEditorContent*
  `public void` **setEditorContent**`( String` **editorContent** `)`
  - **Usage**
    - Sets the current content of the file/editor which was affected by this operation.

      This is a required value.
  - **Parameters**
    - `editorContent` - current editor content, never null.

- *setOffset*
  `public void` **setOffset**`( int` **offset** `)`
  - **Usage**
    - Sets the offset within the document at which the operation occurred.

      This is a required value.

- **Parameters**
  - `offset` - the offset of this event, always >=0.

- *setSelection*
  `public void` **setSelection**`( String` **selection** `)`
  - **Usage**
    - Sets the current selection in the editor.

      This is a required value.
  - **Parameters**
    - `selection` - the current selection, never null.

- *setType*
  `public void` **setType**`( EclipseCutCopyPasteEvent.Type` **type** `)`
  - **Usage**
    - Sets the `EclipseCutCopyPasteEvent.Type` for this event.
      The value may not be `EclipseCutCopyPasteEvent.Type#NULL` .

      This is a required value.
  - **Parameters**
    - `type` - the type for this event, never null.

- *toString*
  `public String` **toString**`( )`

6.1.8 Class **EclipseCutCopyPasteEvent.Type**

The possible types of `EclipseCutCopyPasteEvent` s.

Declaration

public static final class EclipseCutCopyPasteEvent.Type
extends Enum

Fields

- public static final EclipseCutCopyPasteEvent.Type COPY
  - A copy operation.
- public static final EclipseCutCopyPasteEvent.Type CUT
  - A cut operation.
- public static final EclipseCutCopyPasteEvent.Type PASTE
  - A paste operation.
- public static final EclipseCutCopyPasteEvent.Type NULL
  - This is a special activity type which must not be sent to the dispatcher. It's for internal initialisation only.

6.1.9 Class **EclipseEditorPartEvent**

This event is generated by the `CPC Sensor` module, whenever a file is opened or closed in an editor or when an editor window gains or looses the input focus.
The event only contains information about the type of event.

Declaration

public class EclipseEditorPartEvent
extends core.api.hub.event.EclipseEvent

Constructors

- *EclipseEditorPartEvent*
  `public` **EclipseEditorPartEvent**`( String` **user**`, String` **project** `)`
  - **Usage**
    - Creates a new `EclipseEditorPartEvent` for the given user and project.
  - **Parameters**
    - `user` - the current user, never null.
    - `project` - the project for the file affected by this event, never null.

Methods

- *getType*
  `public EclipseEditorPartEvent.Type` **getType**`( )`
  - **Usage**
    - Retrieves the type of this event.
  - **Returns** - the type of this editor part event, never null.

- *isValid*
  `public boolean` **isValid**`( )`

- *setType*
  `public void` **setType**`( EclipseEditorPartEvent.Type` **type** `)`
  - **Usage**
    - Sets the type of this event.

      This is a required value.
  - **Parameters**
    - `type` - the type of this editor part event, never null.

- *toString*
  `public String` **toString**`( )`

6.1.10 Class **EclipseEditorPartEvent.Type**

The type of an `EclipseEditorPartEvent` .

Declaration

public static final class EclipseEditorPartEvent.Type
extends Enum

Fields

- public static final EclipseEditorPartEvent.Type OPENED
  - The file was just opened in an editor window.
- public static final EclipseEditorPartEvent.Type ACTIVATED
  - The editor window for this file just obtained the input focus.
- public static final EclipseEditorPartEvent.Type DEACTIVATED
  - The editor window for this file just lost the input focus.
- public static final EclipseEditorPartEvent.Type CLOSED
  - The editor window for this file was just closed.

6.1.11 Class **EclipseEvent**

Abstract parent class for all CPC Events which are created by Eclipse sensors.

Declaration

public abstract class EclipseEvent
extends core.api.hub.event.CPCEvent

Constructors

- *EclipseEvent*
  `public` **`EclipseEvent`**`( String` **`user`**`, String` **`project`** `)`
  - **Usage**
    - Creates a new `EclipseEvent` instance.
  - **Parameters**
    - `user` - username of current user, never null.
    - `project` - name of project, never null.

Methods

- *getFilePath*
  `public String` **`getFilePath`**`( )`
  - **Usage**
    - Retrieves the project relative file path for this event.
  - **Returns** - path to file, relative to project, never null

- *getProject*
  `public String` **getProject**`( )`
  - **Usage**
    - Retrieves the project name for this event.
  - **Returns** - project name, never null.

- *getUser*
  `public String` **getUser**`( )`
  - **Usage**
    - Retrieves username of the currently logged in user.
  - **Returns** - username of current user, never null.

- *isFileLocatedInWorkspace*
  `public synchronized boolean` **isFileLocatedInWorkspace**`( )`
  - **Usage**
    - Caching convenience method which yields the same result as `CoreFileUtils#isFileLocatedInWorkspace(String, String)` .
      However, the value is only calculated once, on first call of this method and is than reused for all further calls.
  - **Returns** - true if the file exists **and** is located within the workspace, false otherwise.
  - **See Also**
    - `CoreFileUtils.isFileLocatedInWorkspace(String, String)`

- *isSupportedFile*
  `public synchronized boolean` **isSupportedFile**`( )`
  - **Usage**
    - Caching convenience method which yields the same result as `CoreConfigurationUtils#isSupportedFile(String, String)` .
      However, the value is only calculated once, on first call of this method and is than reused for all further calls.

      This method is thread safe.
  - **Returns** - true if the file is a supported source file, false otherwise.
  - **See Also**
    - `CoreConfigurationUtils.isSupportedFile(String, String)`

- *isValid*
  `public boolean` **isValid**`( )`

- *setFilePath*
  `public void` **setFilePath**`( String` **filePath** `)`
  - **Usage**
    - Sets the project relative file path for this event.
  - **Parameters**
    - `filePath` - project relative file path, never null.

- *subToString*
  `protected String` **subToString**`( )`
  - **Usage**
    - Can be called by sub classes in order to obtain a string which can be included in their `toString()` output.
  - **Returns** - string representation of the values of this class, never null.

6.1.12 Class **EclipseFileAccessEvent**

This event is generated by the `CPC Sensor` module, whenever a text file is opened or closed. It does not matter whether the file was opened in an editor (i.e. user opening the file) or in the background (i.e. refactoring or source reformat on the entire project).

Declaration

public class EclipseFileAccessEvent
extends core.api.hub.event.EclipseEvent

Constructors

- *EclipseFileAccessEvent*
  `public` **EclipseFileAccessEvent**`( String` **user**`, String` **project** `)`
  - **Usage**
    - Creates a new `EclipseFileAccessEvent` for the given user and project.
  - **Parameters**
    - `user` - the current user, never null.
    - `project` - the project for the file affected by this event, never null.

Methods

- *getDocument*
  `public IDocument` **getDocument**`( )`
  - **Usage**
    - Retrieves the `IDocument` instance which was created for this file.
  - **Returns** - the document corresponding to the file, never null.
- *getType*
  `public EclipseFileAccessEvent.Type` **getType**`( )`
  - **Usage**
    - Retrieves the type of access (opened/closed).
  - **Returns** - the type of access, never null.
- *isDirty*
  `public boolean` **isDirty**`( )`
  - **Usage**
    - Checks whether the underlying buffer was still dirty when the file was closed.

      This value has no meaning for events which are not of type `Type#CLOSED` .
  - **Returns** - `true` if buffer was dirty, `false` otherwise.
- *isValid*
  `public boolean` **isValid**`( )`
- *setDirty*
  `public void` **setDirty**`( boolean` **dirty** `)`
  - **Usage**

    * Specifies whether the underlying buffer was still dirty when the file was closed.

      This value has no meaning for events which are not of type `Type#CLOSED` .
      This is an optional value, the default value is `false`.
  - **Parameters**
    * `dirty` - `true` if buffer was dirty, `false` otherwise.

- *setDocument*
  `public void` **setDocument**`( IDocument` **document** `)`
  - **Usage**
    * Sets the `IDocument` instance which was created for this file.

      This is a required value.
  - **Parameters**
    * `document` - the document corresponding to the file, never null.

- *setType*
  `public void` **setType**`( EclipseFileAccessEvent.Type` **type** `)`
  - **Usage**
    * Sets the type of access (opened/closed).

      This is a required value.
  - **Parameters**
    * `type` - the type of access, never null.

- *toString*
  `public String` **toString**`( )`

6.1.13 Class **EclipseFileAccessEvent.Type**

The type of an `EclipseFileAccessEvent` .

Declaration

public static final class EclipseFileAccessEvent.Type
extends Enum

Fields

- public static final EclipseFileAccessEvent.Type OPENED
  - The file was just opened.
- public static final EclipseFileAccessEvent.Type CLOSED
  - The file was just closed.

6.1.14 Class **EclipseFileChangeEvent**

This event is generated whenever a file is moved or removed.
Package/folder renames/moves and project renames will also generate corresponding events of this type for all contained files.
This event is only generated for files which are of interest to CPC (see: `CoreConfigurationUtils#isSupportedFile(String)`).

Due to performance considerations this event will not be generated for files which are only modified. If you require knowledge about simple file content modifications, you should register your own `IResourceChangeListener` directly with Eclipse.

Generated by `CPCResourceChangeListener` and by the `IEventHubListener` which consumes `EclipseFileChangeEvent` s and updates the `IStoreProvider` .
See also: `EclipseFileChangeEvent#setPostStoreProviderMoveUpdate(boolean)` .

Declaration

public class EclipseFileChangeEvent
extends core.api.hub.event.EclipseEvent

Constructors

- *EclipseFileChangeEvent*
  `public` **EclipseFileChangeEvent**`( String` **user**`, String` **project** `)`
  - **Usage**
    * Creates a new `EclipseFileChangeEvent` for the given user and project.
  - **Parameters**
    * `user` - the current user, never null.
    * `project` - the project for the file affected by this event, never null.

Methods

- *clone*
  `public Object` **clone**`( )`
  - **Usage**
    * Clones this `EclipseFileChangeEvent` instance.
  - **Exceptions**
    * `CloneNotSupportedException` - never thrown
  - **See Also**
    * `EclipseFileChangeEvent.setPostStoreProviderMoveUpdate(boolean)` (in 6.1.14, page 64)

- *getNewFilePath*
  `public String` **getNewFilePath**`( )`
  - **Usage**
    * Retrieves the new relative path of this file after a move.
      The path is relative to the project returned by `EclipseFileChangeEvent#getNewProject()` .
  - **Returns** - new project relative path after move, NULL if type is not `EclipseFileChangeEvent.Type#MOVED` , never null otherwise.

- *getNewProject*
  `public String` **getNewProject**`( )`

  - **Usage**
    * Retrieves the new project name for this file after a move.
  - **Returns** - new project name after move, NULL if type is not `EclipseFileChangeEvent.Type#MOVED` , never null otherwise.

- *getType*
  `public EclipseFileChangeEvent.Type` **getType**`( )`
  - **Usage**
    * Retrieves the type of this event.
  - **Returns** - the type of this event, never null.

- *isPostStoreProviderMoveUpdate*
  `public boolean` **isPostStoreProviderMoveUpdate**`( )`
  - **Usage**
    * True if this event was generated **after** the `IStoreProvider` has been updated to reflect the new location of the file corresponding to this event.
      By default this value is `false`.

      NOTE: The CPC Sensor will **not** generate events with this value set to `true`.
      It is up to the code which processes `EclipseFileChangeEvent` s and updates the `IStoreProvider` to create a **new event** with this value being set to `true`.
  - **Returns** - `true` if `IStoreProvider` was already updated or `false` if this can not be guaranteed. `false` is also returned for all other event types besides `EclipseFileChangeEvent.Type#MOVED` .

- *isValid*
  `public boolean` **isValid**`( )`

- *setNewFilePath*
  `public void` **setNewFilePath**`( String` **newFilePath** `)`
  - **Usage**
    * Sets the new relative path of this file after a move.
      Must not be set if the type of this event is not `EclipseFileChangeEvent.Type#MOVED` . Required and non-null otherwise.
  - **Parameters**
    * `newFilePath` - new project relative path after move, may be null.

- *setNewProject*
  `public void` **setNewProject**`( String` **newProject** `)`
  - **Usage**
    * Sets the new project name for this file after a move.
      Must not be set if the type of this event is not `EclipseFileChangeEvent.Type#MOVED` . Required and non-null otherwise.
  - **Parameters**
    * `newProject` - new project name after move, may be NULL.

- *setPostStoreProviderMoveUpdate*
  `public void` **setPostStoreProviderMoveUpdate**`( boolean` **postStoreProviderUpdate** `)`
  - **Usage**

    - To be used only by the code which updates the `IStoreProvider` to reflect a file move. Typically an `IEventHubListener` which consumes `EclipseFileChangeEvent` s.

      The code which sets this value to `true` must be able to guarantee that the new event will never be dispatched before the original `EclipseFileChangeEvent` with the value `false` was dispatched.

      NOTE: The new `EclipseFileChangeEvent` should be generated by calling `EclipseFileChangeEvent#clone()` and not by manually copying over all fields.
  - **See Also**
    - `EclipseFileChangeEvent.isPostStoreProviderMoveUpdate()`
    - `EclipseFileChangeEvent.clone()`

- *setType*
  `public void` **setType**`( EclipseFileChangeEvent.Type` **type** `)`
  - **Usage**
    - Sets the type of this event.
  - **Parameters**
    - `type` - the type of this event, never null.

- *toString*
  `public String` **toString**`( )`

6.1.15 Class **EclipseFileChangeEvent.Type**

The type of the `EclipseFileChangeEvent` .

Declaration

public static final class EclipseFileChangeEvent.Type
extends Enum

Fields

- public static final EclipseFileChangeEvent.Type MOVED
  - The file was renamed or moved.
- public static final EclipseFileChangeEvent.Type REMOVED
  - The file was removed.

6.1.16 Class **EclipseResourcePersistenceEvent**

This event is generated by the `CPC Sensor` module, whenever a documents persistence state changes, i.e. when a file is saved or reverted.
This event provides information about the type of persistence event as well as whether the document and whether it is currently open in an editor window.

Declaration

public class EclipseResourcePersistenceEvent
extends core.api.hub.event.EclipseEvent

Constructors

- *EclipseResourcePersistenceEvent*
  `public` **EclipseResourcePersistenceEvent**`( String` **user**`, String` **project** `)`
  - **Usage**
    - Creates a new `EclipseResourcePersistenceEvent` for the given user and project.
  - **Parameters**
    - `user` - the current user, never null.
    - `project` - the project for the file affected by this event, never null.

Methods

- *getDocument*
  `public IDocument` **getDocument**`( )`
  - **Usage**
    - Retrieves the `IDocument` instance which corresponds to this event.
  - **Returns** - the document underlying this event, never null.

- *getType*
  `public EclipseResourcePersistenceEvent.Type` **getType**`( )`
  - **Usage**
    - Retrieves the type of this event.
  - **Returns** - the type of this event, never null.

- *isOpenInEditor*
  `public boolean` **isOpenInEditor**`( )`
  - **Usage**
    - Indicates whether the file corresponding to this event is currently open in an editor window. This can be used to distinguish between automatic background actions and user initiated operations.
  - **Returns** - `true` if the file is currently open in an editor, `false` otherwise.

- *isValid*
  `public boolean` **isValid**`( )`

- *setDocument*
  `public void` **setDocument**`( IDocument` **document** `)`
  - **Usage**
    - Sets the `IDocument` instance which corresponds to this event.

      This is a required value.
  - **Parameters**
    - `document` - the document underlying this event, never null.

- *setOpenInEditor*
  `public void` **setOpenInEditor**`( boolean` **openInEditor** `)`
  - **Usage**
    - Specifies whether the file corresponding to this event is currently open in an editor window.

      This is a required value.
  - **Parameters**
    - `openInEditor` - `true` if the file is currently open in an editor, `false` otherwise.

- *setType*
  `public void` **setType**`( EclipseResourcePersistenceEvent.Type` **type** `)`
  - **Usage**
    - Sets the type of this event.

      This is a required value.
  - **Parameters**
    - `type` - the type of this event, never null.

- *toString*
  `public String` **toString**`( )`

6.1.17 Class **EclipseResourcePersistenceEvent.Type**

The type of the `EclipseResourcePersistenceEvent` .

Declaration

public static final class EclipseResourcePersistenceEvent.Type
extends Enum

Fields

- public static final EclipseResourcePersistenceEvent.Type SAVED
  - The file was saved.
- public static final EclipseResourcePersistenceEvent.Type REVERTED
  - The file was reverted to its prior persisted state.

6.1.18 Class **EclipseTeamEvent**

A special team action event which is generated by repository provider specific CPC sensors whenever files are committed to or updated from the repository.

Declaration

public class EclipseTeamEvent
extends core.api.hub.event.EclipseEvent

Constructors

- *EclipseTeamEvent*
  `public` **EclipseTeamEvent**`( String` **user**`, String` **project** `)`

Methods

- *getNewRevision*
  `public String` **getNewRevision**`( )`
  - **Usage**
    - Retrieves the new revision identifier as provided by the repository provider. May be NULL, if no revision data was provided by the repository provider.
  - **Returns** - new revision identifier for this file version, may be NULL.

- *getOldRevision*
  `public String` **getOldRevision**`( )`
  - **Usage**
    - Retrieves the old revision identifier as provided by the repository provider. May be NULL, if no old revision data was available.
  - **Returns** - old revision identifier for the file before this team action, may be NULL.

- *getType*
  `public EclipseTeamEvent.Type` **getType**`( )`
  - **Usage**
    - Retrieves the type of this event.
  - **Returns** - the type of this event, never null.

- *isValid*
  `public boolean` **isValid**`( )`

- *setNewRevision*
  `public void` **setNewRevision**`( String` **revision** `)`
  - **Usage**
    - Sets the new revision identifier as provided by the repository provider.
  - **Parameters**
    - `revision` - new revision identifier for this file version, may be NULL.

- *setOldRevision*
  `public void` **setOldRevision**`( String` **oldRevision** `)`
  - **Usage**
    - Sets the old revision identifier as provided by the repository provider.
  - **Parameters**
    - `oldRevision` - the old revision identifier for the file before this team action, may be NULL.

- *setType*
  `public void` **setType**`( EclipseTeamEvent.Type` **type** `)`
  - **Usage**
    - Sets the type of this event.
  - **Parameters**
    - `type` - the type of this event, never null.

- *toString*
  `public String` **toString**`( )`

6.1.19 Class **EclipseTeamEvent.Type**

Type for `EclipseTeamEvent` s.

Declaration

public static final class EclipseTeamEvent.Type
extends Enum

Fields

- public static final EclipseTeamEvent.Type COMMIT
  - The file was committed to the repository.
    The revision of the local file will have increased but the content should be unaffected.
- public static final EclipseTeamEvent.Type UPDATE
  - The file was updated from the repository.
    The revision as well as the content of the local file will have been affected.

Chapter 7

Package core.api.hub.registry

Package Contents *Page*

7.1 Interfaces

7.1.1 Interface IEventHubListener

This interface is to be implemented by listeners who want to register callbacks with the `IEventHubRegistry` of the `CPCCorePlugin` in order to receive `CPCEvent` notifications.

Declaration

```
public interface IEventHubListener
```

Methods

- *processEvent*
  `public void` **processEvent**`( CPCEvent` **event** `)`
  - **Usage**
    * The callback function which will be called with a `CPCEvent` if this listener has subscribed for that type of event.
      A listener is guaranteed to only receive events of the types which it has subscribed for.

      A listener **MUST NOT** modify any of the data contained in an event object. The same event object is

reused for all registered listeners. A listener must not keep a reference to the event object beyond the duration of this call.
For performance reasons references to the contents of certain events may be retained (to avoid unnecessary cloning).

If multiple listeners have registered for the same event type, listeners are notified in descending order of their priority. If multiple listeners have the same priority, the order of notification is not specified. Synchronous listeners are notified first. Asynchronous dispatching is delay till the last synchronous listener finished the processing of the event.

Depending on the kind of subscription events are dispatched either synchronously or asynchronously.

If synchronous dispatching was requested events are dispatched synchronous to the thread which generated the event. This means that the generator of this event is blocked until processing of the event has been finished. A listener should thus dispatch any long running work in a separate background job, as it might be blocking the main UI thread.
Another aspect of synchronous dispatching is that if multiple threads are dispatching events, a listener may be executed concurrently with different events.

In asynchronous dispatching mode all events are dispatched from special background dispatching threads. A listener does not have to be thread safe in this mode unless it especially set the `threadsafe` parameter to `true` when registering the listener callback object. Listeners which claimed to be thread safe may receive asynchronous events concurrently from different background dispatching threads. It is up to the `IEventHubRegistry` implementation whether to make use of multiple dispatching threads or not.
Very long running tasks should be executed as background jobs even in this mode. Otherwise other listeners might starve.

IMPORTANT NOTE: In synchronous mode events are dispatched in order of arrival, but new synchronous events may interrupt the normal sequence of events. This means that if any of the registered listeners generates new events of the same type, events may be delivered out of order.
If possible a listener should try to avoid generating events of its own type. Repeated generation of such events may lead to endless loops and starvation of other listeners.
It is the responsibility of the listener to ensure that this handled correctly.

In the asynchronous dispatching mode a new asynchronous event will not be processed until all listeners for the last event were successfully executed. A new event can thus not force its way in between as is possible with synchronous events.

In general asynchronous dispatching should be used whenever possible.

Synchronous listeners should take careful note of the locks which the event generator might be holding. There may be situations which can easily lead into a deadlock scenario if a listener behaves incorrectly. Actions which require synchronisation with the main UI thread tend to be especially dangerous.

– **Parameters**
  ∗ `event` - a new event which the listener may process, never null.

7.1.2 Interface **IEventHubRegistry**

The central event dispatcher for `CPCEvent` s, a key component of the CPC Framework. Each CPC subsystem can send events which may be of interest to other subsystems to this event hub. They are then dispatched to all interested parties.

Events are dispatched synchronously and asynchronously, depending on the preferences of the subscribed listeners.

Programmatical subscription/unsubscription for arbitrary `CPCEvent` types is handled by the `subscribe` and `unsubscribe` methods. Other modules interested in dynamically modifying their subscriptions should use these methods to subscribe/unsubscribe for the event types they are interested in.

However, the **recommended** way for static subscription is the `eventHubListeners` extension point of the `CPC Core` module.

A reference to the currently active `IEventHubRegistry` instance can be obtained via `CPCCorePlugin#getEventHubRegistry()` .

NOTE: Any implementation of this interface also needs to implement `IManagableEventHubRegistry` .

Declaration

```
public interface IEventHubRegistry
```

Methods

- *dispatch*
  `public void` **dispatch**`( CPCEvent` **event** `)`
  - **Usage**
    - Dispatch the event to all interested parties.

      The event needs to be `valid`, otherwise an error is logged and the event is ignored.
      Events are automatically `sealed` once they are passed to this method.
      It is up to the registered listeners whether this event is dispatched synchronously, asynchronously or both.
      The caller is guaranteed to be shielded from any potential exceptions thrown by any of the listeners. An exception thrown by one listener will not affect other listeners.

      The caller should try to release as many locks as possible prior to calling this method as the focus might be passed to some long running synchronous listener. If a caller absolutely must not be blocked by the event dispatching process, it should move the call to this method into a separate background thread. However, even in that case some of the then concurrently executed listeners may contend for the main UI thread.
      Calling this method from the background event dispatching thread is explicitly permitted. The event dispatching order to asynchronous listeners will be correctly maintained.

      An `IStoreProvider` exclusive write lock may only be held during event dispatching, if the specification of the `CPCEvent` specifically states this fact. In all other cases an event generating thread will have to queue events internally and release the `IStoreProvider` lock before actually dispatching them with this method.
  - **Parameters**
    - `event` - the event to dispatch, never null
  - **See Also**
    - `CPCEvent.isValid()`
    - `CPCEvent.seal()`
    - `IEventHubListener`

- *subscribe*
  `public void` **subscribe**`( Class` **eventType**`, boolean` **synchronous**`, byte` **priority**`, IEventHubListener` **listener** `)`
  - **Usage**

    * Registers a listener callback to receive `CPCEvent` notifications. The class hierarchy is taken into account, i.e.
      `subscribe(CPCEvent.class, this);`
      will subscribe to all events.
      If a specific event class is provided, then the listener will only be registered for that event type.
  - **Parameters**
    * `eventType` - the `CPCEvent` subclass for which the listener should receive notifications, never null.
    * `synchronous` - `true` if this listener expects events to be dispatched in a synchronous fashion. In this case the sender of the event will be blocked until all synchronous listeners have processed the event. If this is `false` the events are dispatched in a separate background thread.
    * `priority` - sorting attribute which affects the order in which multiple listeners registered for the same event type will be notified. Listeners are called in descending order of their priority. If multiple listeners have the same priority, the order in not specified.
      Values may be negative, a good default value is `0`.
    * `listener` - the listener callback, never null.

- *unsubscribe*
  `public boolean` **unsubscribe**`( Class` **eventType**`, boolean` **synchronous**`, IEventHubListener` **listener** `)`
  - **Usage**
    * Unregisters a `CPCEvent` listener callback.
      A programmatically subscribed listener, should always be unsubscribed once it is no longer needed.
      Unsubscribing of listeners which were registered via the `eventHubListeners` extension point is not required.
  - **Parameters**
    * `eventType` - the `CPCEvent` subclass for which the listener was originally registered, never null
    * `synchronous` - should be set to the same value as was used during subscription of this listener.
    * `listener` - the listener to remove, never null
  - **Returns** - true if the listener was registered, false if listener was unknown

7.1.3 Interface **IManagableEventHubRegistry**

Management extension to the `IEventHubRegistry` interface.

All `IEventHubRegistry` implementation need to implement this interface too.

Methods of this interface may only be used by the `CPC Core` module.

Declaration

public interface IManagableEventHubRegistry
implements IEventHubRegistry

Methods

- *shutdown*
  `public void` **shutdown**`( )`
  - **Usage**

∗ Called when the event hub registry is being shut down.
  This typically only happens when the Eclipse IDE is being shutdown.

  A registry implementation should not depend on a call to this method as there might be shutdown scenarios in which the `CPC Core` module is unable to call this method in time.

Chapter 8

Package core.api.provider

Package Contents *Page*

8.1 Interfaces

8.1.1 Interface **IManagableProvider**

Special extension interface for `IProvider` which provides internal life cycle management methods.

This interface lists provider instance management related methods which must only be called by the active `IProviderRegistry` implementation.

All `IProvider` implementations must also implement this interface.
Their corresponding API interfaces should **not** extend this interface.

Declaration

public interface IManagableProvider **implements** IProvider

Methods

- *onLoad*
  `public void` **onLoad**`( )`
  - **Usage**

    * Called when this provider is first returned to a user by the `IProviderRegistry` .
      The method will be called only once per provider instance.

      If this provider was registered as a singleton, subsequent requests to `IProviderRegistry#lookupProvider(Class)` will always return the same instance and `onLoad()` will **not** be called again.

      In other words, it is guaranteed that this method is called exactly once before this instance is first used.

      Clients must not call this method.

- *onUnload*
  `public void` **onUnload**`( )`
  - **Usage**
    * Called when this provider is unregistered with the `IProviderRegistry` .
      It is guaranteed that this provider will not be used again once this method was called.

      For a provider instance which has not been registered as a singleton this method will be called once the client indicates that it no longer needs this provider. However, a client is **not** required to do so!

      Providers should **not** depend on this method for their correct operation. There may be shutdown scenarios in which the `IProviderRegistry` will not be able to unregister all providers in time. And there may be situations in which clients are not notifying the provider registry about no longer needed provider instances.
      A provider instance may thus be garbage collected without ever receiving a call to this method.

      Clients must not call this method.

8.1.2 Interface **IPoolableProvider**

A special extension interface of `IManagableProvider` for service providers which want to request instance pooling.

A provider which prefers instance pooling over the creation of a new instance for every client lookup should implement this interface.
Whenever possible it is recommended for providers to be registered as singletons.

An `IProviderRegistry` implementation is **not** required to support provider instance pooling. If pooling is not supported, providers implementing this interface are handled like normal `IManagableProvider` s.

Declaration

public interface IPoolableProvider
implements IManagableProvider

Methods

- *addedToPool*
  `public void` **addedToPool**`( )`
  - **Usage**

    - Called when a new provider instance is first added to an instance pool.
      This method is guaranteed to be called before `#leavingPool()` and `#removedFromPool()` .

      A provider implementation can use a call to this method as indication of pooling support in the current provider registry. If the provider is used before this method is called, pooling is not supported.

- *leavingPool*
  `public void` **leavingPool**`( )`
  - **Usage**
    - Called shortly before this instance is handed out to a client.

- *removedFromPool*
  `public void` **removedFromPool**`( )`
  - **Usage**
    - Called shortly before provider instance is removed from the pool and discarded.
      This typically happens on shutdown or when the provider registry decides that there are too many unused instances in the pool.

- *returningToPool*
  `public void` **returningToPool**`( )`
  - **Usage**
    - Called shortly after a client has indicated that it no longer needs this instance and the instance is about to be returned to the pool.

8.1.3 INTERFACE **IProvider**

General provider interface implemented by all service providers, used to allow loose coupling of CPC subsystems.

The different CPC subsystems can provide or require specific providers. The `IProviderRegistry` of the `CPCCorePlugin` is used as a central registry to register and obtain providers.

All `IProvider` implementations need to provide a zero argument constructor.

If the provider interface specification does not explicitly state otherwise, all providers need to be thread safe.

NOTE: All implementations of this interface also have to implement `IManagableProvider` . However, the API interfaces should only extend `IProvider` . The methods of `IManagableProvider` are only meant for internal use by the `IProviderRegistry` .

DECLARATION

```
public interface IProvider
```

METHODS

- *getProviderName*
  `public String` **getProviderName**`( )`
  - **Usage**

    * Retrieves the name of this provider.

      Used in configuration dialogs to identify the provider (if it is already loaded, otherwise the name from the plugin.xml will be used, it is probably a good idea to keep the two names equal at all times). Also used for debug output.
  - **Returns** - a short string describing this provider, never null.

- *toString*
  `public String toString( )`
  - **Usage**
    * For debugging purposes all provider authors are encouraged to provide a meaningful toString method.

Chapter 9

Package core.api.provider.classification

Package Contents *Page*

9.1 Interfaces

9.1.1 Interface IClassificationProvider

The CPC API for clone classification providers.
A classification provider takes a clone objects, analyses it and attaches a number of classifications to it.
Classifications are Strings which usually correspond to the `CLASSIFICATION_*` constants of this class.

3rd parties may add their own classification strings. Such strings need to have a globally unique prefix to prevent collisions with other classifications.
The prefix "`cpc.`" is reserved for the default CPC classifiers.
Classification strings may only contain letters, numbers and dots. Classification strings are case sensitive.

Declaration

public interface IClassificationProvider
implements core.api.provider.IProvider

Fields

- public static final String CLASSIFICATION_CLASS

  - The clone contains at least one complete java class.

- public static final String CLASSIFICATION_METHOD
  - The clone contains at least one complete java method.
- public static final String CLASSIFICATION_LOOP
  - The clone contains at least one complete loop construct.
- public static final String CLASSIFICATION_CONDITION
  - The clone contains at least one complete java condition block.
    I.e. a complete "if () { ... } else { ... }" construct.
- public static final String CLASSIFICATION_IDENTIFIER
  - The clone contains a complete identifier and nothing else. Whitespaces and comments are ignored.
- public static final String CLASSIFICATION_COMMENT
  - The clone contains only comments and whitespaces or a part of a comment.
- public static final String CLASSIFICATION_COMPLEX
  - The clone contains potentially complex code.
    A clone should be tagged with this classification if it seems likely that the clone is non-trivial and that any update anomalies inside such a clone are potentially interesting candidates for CPC Warnings.
- public static final String CLASSIFICATION_TEMPLATE
  - The clone contains is probably a template code fragment.
    Clones of this kind can be similar to each other, but the underlying semantics are usually not related.
    This classification should only be set, if there is a high probability that the decision is correct. Other modules may base their decisions on this fact. I.e. CPC Notify may decide to ignore a clone modification if this classification is set.
    However, if it is absolutely clear that there is no point in tracking this clone at all. The classification provider should return a `Result#REJECTED` result.

Methods

- *classify*
  `public IClassificationProvider.Result` **classify**`( IClassificationProvider.Type` **type**`, ICloneFile` **cloneFile**`, IClone` **clone**`, String` **fileContent**`, IClone` **originClone** `)`
  - **Usage**
    * Takes a clone object and passes it to all registered classification strategies to decide on the correct classifications.

      The new classifications are directly added to the clones classifications data structure (the clone object is updated in place).
      It is up to the specified `type` and the implementation how existing classification are handled.

      Providing the file content is optional, however, if it is already present for some reason, it should be provided to reduce load.

      A classification provider may try to obtain additional information for the corresponding file from the Eclipse environment, if it is running. I.e. a classification provider may try to obtain the AST for a Java class.
      It is up to the classification provider implementation whether to make use of any such additional information or not.

- **Parameters**
  - `type` - the type of classification to be performed, never null.
  - `cloneFile` - the clone file which contains the given clone, never null.
  - `clone` - the clone to classify, never null.
  - `fileContent` - current content of the file this clone is located in, may be NULL in which case the content will be retrieved from an open editor or the filesystem, when needed.
  - `originClone` - optional reference to the `IClone` which the given `clone` was copied from. This will usually only be available for `Type#INITIAL` calls and even then it is optional. May be NULL.
- **Returns** - the general classification result `Result` , never null.

9.2 Classes

9.2.1 Class **IClassificationProvider.Result**

Possible results of the `#classify(Type, ICloneFile, IClone, String, IClone)` method.

Declaration

```
public static final class IClassificationProvider.Result
extends Enum
```

Fields

- public static final IClassificationProvider.Result ACCEPTED
  - The classifier classified this clone as being suitable for tracking.
- public static final IClassificationProvider.Result REJECTED
  - The classifier classified this clone as NOT being suitable for tracking.
    This clone should simply be ignored.
- public static final IClassificationProvider.Result ERROR
  - Returned if any non-recoverable error occurs during clone classification.
    This indicates that no classification data was added to the clone.
    Users of this API will usually not have to check for this condition, it can be handled similarly to `Result#ACCEPTED` in most situations.

9.2.2 Class **IClassificationProvider.Type**

Specifies the type of classification to be performed.

Declaration

```
public static final class IClassificationProvider.Type
extends Enum
```

Fields

- public static final IClassificationProvider.Type INITIAL
  - First classification of a newly created clone.
- public static final IClassificationProvider.Type INCREMENTAL
  - Incremental update of classifications of an existing clone which may or may not have been classified before.
    In this mode old classifications may be preserved if this reduces the processing effort required for the classification of the clone or if some of the classifications are not intended to be recalculated after clone creation.

    It is up to the `IClassificationProvider` whether this state is actually handled differently from `Type#RECLASSIFY` .
- public static final IClassificationProvider.Type RECLASSIFY
  - Complete reclassification of the clone.
    All existing classifications are removed.

Chapter 10

Package core.api.provider.cpcrepository

Package Contents *Page*

10.1 Interfaces

10.1.1 Interface **ICPCRepositoryProvider**

An `ICPCRepositoryProvider` provides a centralised remote storage service for CPC clone data which may be access concurrently from multiple CPC installations.

A typical use for a provider of this type is to store the current cpc data for a file in a central location, whenever it is committed into a source repository and to fetch the most up to date cpc data for a file whenever it is checked out, updated or merged.

Most methods of this API are likely to require remote calls. They can thus fail and are potentially long running.

NOTE: Any implementation must guarantee that concurrent calls to the put and get methods on this and other systems do not result in invalid data.

Declaration

public interface ICPCRepositoryProvider
implements core.api.provider.IProvider

Methods

- *createRevision*
  `public ICPCRevision` **createRevision**`( )`
  - **Usage**

    - Creates a new and empty `ICPCRevision` instance which can then be filled by the client.

      `ICPCRevision` instances must not be mixed between `ICPCRepositoryProvider` s and a client may not use its own implementations.
  - **Returns** - new and empty `ICPCRevision` instance, never null.

- *getRevision*
  `public ICPCRevision` **getRevision**`( String` **revisionId**`, String` **cloneFileUuid** `)`
  - **Usage**
    - Retrieves the cpc data revision with the given `revisionId` for the `ICloneFile` with the given `cloneFileUuid`.
  - **Parameters**
    - `revisionId` - the revision identifier to retrieve, never null.
    - `cloneFileUuid` - the `ICloneFile#getUuid()` value of the file to retrieve the data for, never null.
  - **Returns** - a `ICPCRevision` or NULL if no such revision was found.

- *getRevision*
  `public ICPCRevision` **getRevision**`( String` **revisionId**`, String` **project**`, String` **filePath** `)`
  - **Usage**
    - Retrieves the cpc data revision with the given `revisionId` for the `ICloneFile` with the given `project` and `filePath` location.
  - **Parameters**
    - `revisionId` - the revision identifier to retrieve, never null.
    - `project` - the project name of the `ICloneFile` to retrieve the data for, never null.
    - `filePath` - the project relative path of the `ICloneFile` to retrieve the data for, never null.
  - **Returns** - a `ICPCRevision` or NULL if no such revision was found.

- *hintEndTransaction*
  `public void` **hintEndTransaction**`( )`
  - **Usage**
    - This method must only be called if `#hintStartTransaction()` was called. For each call to `#hintStartTransaction()` there needs to be exactly one call to `#hintEndTransaction()` .

      A typical use of this method is the shutdown of some remote network connection.
  - **See Also**
    - `ICPCRepositoryProvider.hintStartTransaction()`

- *hintStartTransaction*
  `public void` **hintStartTransaction**`( )`
  - **Usage**
    - This method should be called by clients of this interface if it is expected that multiple repository operations will be done within a short period of time.

      It is up to the implementation of this interface whether to make use of this additional information. The implementation may not rely on a call to this method. The visible behaviour of the implementation must not change depending on whether this method is called or not.

      While calling this method is optional, a client must call `#hintEndTransaction()` once it called this method.

      NOTE: this method is only meant to improve performance. No transactional properties are guaranteed by this API.

      A typical use of this method is the setup of some remote network connection.

- **See Also**
  - `ICPCRepositoryProvider.hintEndTransaction()`

- *isAvailable*
  `public boolean` **isAvailable**`( )`
  - **Usage**
    - Checks whether this cpc repository is currently available.

      This method will typically try to connect to a remote location and verify that there are no connectivity or version incompatibility issues which would prevent the normal use of this provider.

      If this method returns `false` most other methods of this API are likely to throw an exception when used. However return value of `true` does not guarantee that no exception will occur.
  - **Returns** - `true` if the repository is ready for use, `false` if there are any conditions which prevent normal use.

- *purgeRevision*
  `public boolean` **purgeRevision**`( String` **revisionId**`, String` **cloneFileUuid** `)`
  - **Usage**
    - Tells the `ICPCRepositoryProvider` that the specified revision is no longer needed and can be deleted. The `ICPCRepositoryProvider` may not return a revision to any client once it has been marked for purging in this way.
  - **Parameters**
    - `revisionId` - the revision identifier of the revision to purge, never null.
    - `cloneFileUuid` - cloneFileUuid the `ICloneFile#getUuid()` value of the file to purge the revision for, never null.
  - **Returns** - `true` if such a revision was found and purged, `false` if no such revision existed.

- *purgeRevision*
  `public boolean` **purgeRevision**`( String` **revisionId**`, String` **project**`, String` **filePath** `)`
  - **Usage**
    - Tells the `ICPCRepositoryProvider` that the specified revision is no longer needed and can be deleted. The `ICPCRepositoryProvider` may not return a revision to any client once it has been marked for purging in this way.
  - **Parameters**
    - `revisionId` - the revision identifier of the revision to purge, never null.
    - `project` - the project name of the `ICloneFile` to purge the data for, never null.
    - `filePath` - the project relative path of the `ICloneFile` to purge the data for, never null.
  - **Returns** - `true` if such a revision was found and purged, `false` if no such revision existed.

- *putRevision*
  `public void` **putRevision**`( ICPCRevision` **cpcRevision** `)`
  - **Usage**
    - Stores the given cpc data revision in the remote repository.
      An exception will be thrown if the repository already contains an entry for that file with the same revision identifier.
  - **Parameters**
    - `cpcRevision` - the cpc data revision to store, never null.

10.1.2 Interface **ICPCRevision**

A `ICPCRevision` is a simple wrapper object for remotely stored clone data packages.
It contains:

- a revision id string
- an `ICloneFile` instance
- a list of `IClone` instances for the file

`ICPCRevision` instances are used in combination with `ICPCRepositoryProvider` operations.
A new instance for this interface can be obtained via: `ICPCRepositoryProvider#createRevision()`

Declaration

```
public interface ICPCRevision
```

Methods

- *getCloneFile*
  `public ICloneFile` **getCloneFile**`( )`
  - **Usage**
    - Retrieves the `ICloneFile` instance for this cpc revision.
  - **Returns** - an `ICloneFile` instance, never null.

- *getClones*
  `public List` **getClones**`( )`
  - **Usage**
    - Retries a list of `IClone` instances which are part of this revision.
      They are all located with the `ICPCRevision#getCloneFile()` file.
  - **Returns** - a list of `IClone` for this file, never null.

- *getRevisionId*
  `public String` **getRevisionId**`( )`
  - **Usage**
    - Retrieves the revision identifier string for this cpc revision.
      I.e. a revision identifier as it is assigned to source files from the main source repository provider.
  - **Returns** - revision identifier, never null.

- *isValid*
  `public boolean` **isValid**`( )`
  - **Usage**
    - Checks whether all required fields for this element have been set.
  - **Returns** - true if all required fields have been set, false otherwise.

- *setCloneFile*
  `public void` **setCloneFile**`( ICloneFile` **cloneFile** `)`
  - **Usage**
    - Sets the `ICloneFile` instance for this cpc revision.
  - **Parameters**
    - `cloneFile` - an `ICloneFile` instance, never null.

- **See Also**
  - `ICPCRevision.getCloneFile()`

- *setClones*
  `public void` **setClones**`( List` **clones** `)`
  - **Usage**
    - Specifies a list of `IClone` instances which are part of this revision.
  - **Parameters**
    - `clones` - a list of `IClone` for this file, never null.
  - **See Also**
    - `ICPCRevision.getClones()`

- *setRevisionId*
  `public void` **setRevisionId**`( String` **revisionId** `)`
  - **Usage**
    - Sets the revision identifier string for this cpc revision.
  - **Parameters**
    - `revisionId` - revision identifier, never null.
  - **See Also**
    - `ICPCRevision.getRevisionId()`

- *toString*
  `public String` **toString**`( )`
  - **Usage**
    - All implementations should provide a meaningful toString() method for debugging purposes.

Chapter 11

Package core.api.provider.data

11.1 Interfaces

11.1.1 Interface **ICloneFactoryProvider**

Public clone data object factory provider API.
A clone factory provider is used by all CPC modules for the creation of clone objects. Clone objects are never created directly, circumventing the clone factory provider.

There may only be one active clone factory provider at all times.

3rd party extensions should register their own `ICloneObject` , `ICloneObjectSupport` and `ICloneObjectExtension` classes with the clone factory provider via the extension point:
`core.cloneDataElements`

Declaration

public interface ICloneFactoryProvider
implements core.api.provider.IProvider

Methods

- *getInstance*
  `public ICloneDataElement` **getInstance**`( Class` **type** `)`
  - **Usage**
    * Creates a new instance of the specified `ICloneDataElement` sub class.
      For `ICloneObject` sub classes a new unique uuid is automatically generated.

      Valid values for `type` are:
      · IClone.class
      · ICloneFile.class
      · ICloneGroup.class

        · ICloneAnnotation.class
        · IClonePosition.class
        · any of the currently used implementations of those interfaces
        · any other registered `ICloneObject` , `ICloneObjectSupport` or `ICloneObjectExtension`

- **Parameters**
  - `type` - the `ICloneDataElement` sub class to create a new instance for, never null.
- **Returns** - a new instance which is guaranteed to be castable to the specified `type` or null if no such `ICloneDataElement` sub class is available.

- *getInstance*
  `public ICloneObject` **getInstance**`( Class` **type**`, String` **uuid** `)`
  - **Usage**
    - Creates a new instance of the specified `ICloneObject` sub class.

      This method can **not** be used to create instances for `ICloneObjectSupport` sub-interfaces.

      Valid values for `type` are:
        · IClone.class
        · ICloneFile.class
        · ICloneGroup.class
        · ICloneAnnotation.class
        · as well as the currently used implementations of those interfaces
  - **Parameters**
    - `type` - the `ICloneObject` sub class to create a new instance for, never null.
    - `uuid` - the unique uuid to use for the newly created instance.
  - **Returns** - a new instance which is guaranteed to be castable to the specified `type` or null if no such `ICloneObject` sub class is available.

- *getInstanceByPersistenceClassIdentifier*
  `public IStatefulObject` **getInstanceByPersistenceClassIdentifier**`( String` **persistenceClassIdentifier** `)`
  - **Usage**
    - Creates a new instance of a registered `IStatefulObject` for the given `IStatefulObject#getPersistenceClassIdentifier()` value.
  - **Parameters**
    - `persistenceClassIdentifier` - the `IStatefulObject` persistence class identifier to create an instance of an implementation class for, never null.
  - **Returns** - a new instance which is guaranteed to yield `persistenceClassIdentifier` for `IStatefulObject#getPersistenceClassIdentifier()` or NULL if no such class is available.

- *getRegisteredCloneObjectExtensionObjects*
  `public List` **getRegisteredCloneObjectExtensionObjects**`( Class` **parentType** `)`
  - **Usage**
    - Some users of the `getRegistered...` methods need to create temporary instances of the classes during their processing. This adds overhead which can be critical. An extreme case of this is the `IStoreProvider` .
      This method mirrors `ICloneFactoryProvider#getRegisteredCloneObjectExtensions(Class)` but returns shared instances of the extensions instead of their classes.

      IMPORTANT: the returned objects are **shared**. Do **not** modify them in any way.
  - **Parameters**
    - `parentType` - the `ICloneObject` type for which all registered extensions should be returned, never null.
  - **Returns** - a list of **shared** instances of registered clone object extensions for the given parent type, never null.

- *getRegisteredCloneObjectExtensions*
  `public List` **getRegisteredCloneObjectExtensions**`( )`
  - **Usage**
    - Retrieves a list of all registered `ICloneObjectExtension` implementations.
      Implementations are registered with the clone factory provider via the corresponding extension point.

      The returned list and it's elements may not be modified.
  - **Returns** - a list of all registered clone object extensions, never null.

- *getRegisteredCloneObjectExtensions*
  `public List` **getRegisteredCloneObjectExtensions**`( Class` **parentType** `)`
  - **Usage**
    - Same as `ICloneFactoryProvider#getRegisteredCloneObjectExtensions()` but only returns the extensions registered for the given `ICloneObject` type.
  - **Parameters**
    - `parentType` - the `ICloneObject` type for which all registered extensions should be returned, never null.
  - **Returns** - a list of registered clone object extensions for the given parent type, never null.

- *getRegisteredCloneObjects*
  `public List` **getRegisteredCloneObjects**`( )`
  - **Usage**
    - Retrieves a list of all registered `ICloneObject` sub-interface implementations.
      Implementations are registered with the clone factory provider via the corresponding extension point.

      The returned list and it's elements may not be modified.
  - **Returns** - a list of registered clone objects, never null.

- *getRegisteredCloneObjectSupports*
  `public List` **getRegisteredCloneObjectSupports**`( )`
  - **Usage**
    - Retrieves a list of all registered `ICloneObjectSupport` sub-interface implementations.
      Implementations are registered with the clone factory provider via the corresponding extension point.

      The returned list and it's elements may not be modified.
  - **Returns** - a list of registered clone objects, never null.

Chapter 12

Package core.api.provider.merge

Package Contents *Page*

12.1 Interfaces

12.1.1 Interface **IMergeProvider**

A part of the Remote Store API, a merge provider takes local and remote clone data and tries to reconcile any conflicts by merging the clone data to correctly reflect the new contents of the corresponding source file.

Source files are **not** merged by an `IMergeProvider` . This is left to the normal Eclipse procedures. By the time the merge provider is called, the result of the source file merge is already available.

A `IMergeProvider` must not have any side effects. It must neither access the `IStoreProvider` nor the workspace resources in any way.

Declaration

public interface IMergeProvider
implements core.api.provider.IProvider

Methods

- *createTask*
  `public IMergeTask` **createTask**`( )`
  - **Usage**
    - Creates a new, empty `IMergeTask` instance which can then be filled with all the required data to descripe the merge task.

      The returned `IMergeTask` is not yet ”valid”. It must not be passed to `IMergeProvider#merge(IMergeTask)` until all required fields have been set.
  - **Returns** - new, empty `IMergeTask` instance, never null.
  - **See Also**
    - `IMergeTask`

- *merge*
  `public IMergeResult` **merge**`( IMergeTask` **mergeTask** `)`
  - **Usage**
    - Merges local and remote clone data to reflect the new contents of the corresponding source file.

      The concrete merging procedure is left to the implementation. A caller must not make any assumptions about any properties of the merge process.
      It is up to the implementation to decide whether to attempt a Three-Way merge. The presence of the required data in the `IMergeTask` does not guarantee that a Three-Way merge will be executed.
  - **Parameters**
    - `mergeTask` - the merge task descriptor for this operation, never null.
  - **Returns** - a merge result descriptor, never null.
  - **Exceptions**
    - `IllegalArgumentException` - if the provided `IMergeTask` is not ”valid”.
    - `MergeException` - if any errors occur during the merge process.
  - **See Also**
    - `IMergeTask`
    - `IMergeResult`

12.1.2 Interface **IMergeResult**

A result wrapper object for the `IMergeProvider` .

Declaration

```
public interface IMergeResult
```

Methods

- *getCloneFile*
  `public ICloneFile` **getCloneFile**`( )`
  - **Usage**
    - The new `ICloneFile` data for the merged file.

  - **Returns** - merged `ICloneFile` , never null.

- *getLocalPerspective*
  `public IMergeResultPerspective` **getLocalPerspective**`( )`
  - **Usage**
    - Description of the merge implications from the perspective of the local clone data.

      This is the perspective needed to update the local `IStoreProvider` .
  - **Returns** - local perspective of the merge implications, never null.
  - **See Also**
    - `IMergeResultPerspective`

- *getMergedClones*
  `public List` **getMergedClones**`( )`
  - **Usage**
    - A list of the final `IClone` instances for the merged source file.
      The list does not contain duplicates.

      This data might be calculated on demand. A call might therefore be expensive.
      This method is not guaranteed to be thread save.

      This is equivalent to (after removing duplicates):
      `getAddedClones()+getMovedClones()+getModifiedClones()+getUnchangedClones()`

      A client which only intents to update an `IStoreProvider` will not need this information.

      The order of the `IClone` instances in this list is not defined.
  - **Returns** - a complete list of `IClone` instances for the final merged source file, may be empty, never null.

- *getRemotePerspective*
  `public IMergeResultPerspective` **getRemotePerspective**`( )`
  - **Usage**
    - Description of the merge implications from the perspective of the remote clone data.

      In most cases a client will probably only need the `IMergeResult#getLocalPerspective()` or `IMergeResult#getMergedClones()` .
  - **Returns** - remote perspective of the merge implications, never null.
  - **See Also**
    - `IMergeResultPerspective`

- *getStatus*
  `public IMergeResult.Status` **getStatus**`( )`
  - **Usage**
    - Information about the success or failure of this merge operation.
  - **Returns** - status of the merge operation, never null.
  - **See Also**
    - `IMergeResult.Status`

- *isFullyMerged*
  `public boolean` **isFullyMerged**`( )`
  - **Usage**

  - ∗ Checks whether this result represents a fully merged state.
    Convenience method.
  - **Returns** - true if `IMergeResult#getStatus()` is `Status#FULL_MERGE` .

- *toString*
  `public String` **`toString`**`( )`
  - **Usage**
    - ∗ All implementations should provide a meaningful toString() method for debugging purposes.

12.1.3 Interface **IMergeResultPerspective**

An `IMergeResultPerspective` describes the changes made during an `IMergeProvider` merge of local and remote clone data from either the local or the remote perspective.

It can be thought of as a "diff" which can be applied to the former clone data of the corresponding "side" and which will then yield the new merged clone data.

The resulting clone data for both "sides" will always be the same. But depending on the perspective a clone may fall into different "change categories".

Some examples:
- A clone which was created on this side will fall into the "unchanged" (or maybe "moved", depending on merge) category on this side and into the "added" category on the other side.
- A clone which changed its position due to editing of the document on the other side (and for which the position remained unchanged on this side) will fall into the "moved" category on this side and into the "unchanged" (or maybe "moved", depending on merge) category on the other side.

Declaration

```
public interface IMergeResultPerspective
```

Methods

- *getAddedClones*
  `public List` **`getAddedClones`**`( )`
  - **Usage**
    - ∗ A list of new clones which were added due to actions on the other "side".

      Due to the uniqueness of clone UUIDs a newly added clone can't be part of both perspectives.

      The order of the `IClone` instances in this list is not defined.
  - **Returns** - a list of `IClone` instances, may be empty, never null.

- *getLostClones*
  `public List` **`getLostClones`**`( )`
  - **Usage**
    - ∗ A list of former clones of this "side" which were dropped due to merge conflicts.

      Lost clones which existed on both sides will be part of both perspectives.

      The order of the `IClone` instances in this list is not defined.

  - **Returns** - a list of `IClone` instances, may be empty, never null.

- *getModifiedClones*
  `public List` **getModifiedClones**`( )`
  - **Usage**
    - A list of former clones of this ”side” for which the **content** was modified due to actions on the other ”side”.

      The order of the `IClone` instances in this list is not defined.
  - **Returns** - a list of `IClone` instances, may be empty, never null.
  - **See Also**
    - `CloneModificationEvent.getModifiedClones()`

- *getMovedClones*
  `public List` **getMovedClones**`( )`
  - **Usage**
    - A list of former clones of this ”side” which were moved due to actions on the other ”side”.
      This also includes clone instances which had any other values (beside the content) modified, i.e. extension data.

      The order of the `IClone` instances in this list is not defined.
  - **Returns** - a list of `IClone` instances, may be empty, never null.
  - **See Also**
    - `CloneModificationEvent.getMovedClones()`

- *getName*
  `public String` **getName**`( )`
  - **Usage**
    - Retrieves a human readable name for this perspective.
      By default this is either ”`local`” or ”`remote`”.
  - **Returns** - the name for this perspective, never null.

- *getRemovedClones*
  `public List` **getRemovedClones**`( )`
  - **Usage**
    - A list of former clones of this ”side” which were removed due to user actions on the other ”side”.

      Clones which were removed on both sides will be part of both perspectives.

      The order of the `IClone` instances in this list is not defined.
  - **Returns** - a list of `IClone` instances, may be empty, never null.

- *getUnchangedClones*
  `public List` **getUnchangedClones**`( )`
  - **Usage**
    - A list of former clones of this ”side” which were not affected by this merge.

      These clones are always part of both perspectives.

      The order of the `IClone` instances in this list is not defined.
  - **Returns** - a list of `IClone` instances, may be empty, never null.

12.1.4 Interface **IMergeTask**

A task description object for the `IMergeProvider` .

An instance can be obtained via `IMergeProvider#createTask()` .

Some values are required, some values are optional.
Depending on the provided data a Two-Way or a Three-Way merge may be executed by the merge provider.

The contents of a `IMergeTask` must not be modified in any way once it has been completely filled with data.

Rationale:

> Depending on the persistence provider used, it may not always be possible for a CPC Sensor and CPC Remote Store provider to obtain all the data needed for a Three-Way merge.
> To allow maximum flexibility an `IMergeProvider` must thus be able to handle Two-Way merges if the data required for a Three-Way merge can not be provided.

Declaration

```
public interface IMergeTask
```

Methods

- *isValid*
  `public boolean` **isValid**`( )`
  - **Usage**
    - Checks whether this task is valid.
  - **Returns** - true if all required fields have been set, false otherwise.

- *setBaseCloneFile*
  `public void` **setBaseCloneFile**`( ICloneFile` **cloneFile** `)`
  - **Usage**
    - The common base `ICloneFile` instance.

      Optional value.
  - **Parameters**
    - `cloneFile` - common base clone file instance, may be NULL.

- *setBaseClones*
  `public void` **setBaseClones**`( List` **clones** `)`
  - **Usage**
    - The common base `IClone` instances.

      Optional value.
  - **Parameters**
    - `clones` - common base clones, may be empty, may be NULL.

- *setBaseSourceFileContent*
  `public void` **setBaseSourceFileContent**`( String` **content** `)`

  - **Usage**
    - The common base source file content.

      Optional value.
  - **Parameters**
    - `content` - common base content of the source file, may be NULL.

- *setLocalBaseInSyncHint*
  `public void` **setLocalBaseInSyncHint**`( boolean` **localBaseInSyncHint** `)`
  - **Usage**
    - Specifies whether the current revision is in sync with the base revision.

      In some cases no information about the base revision might be available. However, the caller may still be able to determine whether the current local clone data was potentially modified or whether it can be guaranteed to be in sync with the base revision.

      The default value is `false` which indicates that nothing is known about the synchronisation status between the local and the base revision.

      If the caller can guarantee that the local clone data was not modified and that thus no merge is needed, this value should be set to `true`.

      If this value is `true` a merge provider may simply "overwrite" the local clone data with the remote clone data without merging.
  - **Parameters**
    - `localBaseInSyncHint` - `False` (default) if local and base revision might differ. `True` if local and base revision are guaranteed to be in sync.

- *setLocalCloneFile*
  `public void` **setLocalCloneFile**`( ICloneFile` **cloneFile** `)`
  - **Usage**
    - The local `ICloneFile` instance (before the merge).

      Required value.
  - **Parameters**
    - `cloneFile` - old local clone file instance, never null.

- *setLocalClones*
  `public void` **setLocalClones**`( List` **clones** `)`
  - **Usage**
    - The local `IClone` instances (before the merge).

      Required value.
  - **Parameters**
    - `clones` - old local clones, may be empty, never null.

- *setLocalSourceFileContent*
  `public void` **setLocalSourceFileContent**`( String` **content** `)`
  - **Usage**
    - The local source file content before the merge.

      Required value.

  - **Parameters**
    - `content` - old local content of the source file, never null.

- *setMergedSourceFileContent*
  `public void` **setMergedSourceFileContent**`( String` **content** `)`
  - **Usage**
    - The result of the merge of the two source files.

      Required value.
  - **Parameters**
    - `content` - the content of the new source file on disk, never null.

- *setRemoteCloneFile*
  `public void` **setRemoteCloneFile**`( ICloneFile` **cloneFile** `)`
  - **Usage**
    - The remote `ICloneFile` instance (before the merge).

      Required value.
  - **Parameters**
    - `cloneFile` - old remote clone file instance, never null.

- *setRemoteClones*
  `public void` **setRemoteClones**`( List` **clones** `)`
  - **Usage**
    - The remote `IClone` instances (before the merge).

      Required value.
  - **Parameters**
    - `clones` - old remote clones, may be empty, never null.

- *setRemoteSourceFileContent*
  `public void` **setRemoteSourceFileContent**`( String` **content** `)`
  - **Usage**
    - The remote source file content before the merge.

      Required value.
  - **Parameters**
    - `content` - old remote content of the source file, may be NULL.

- *toString*
  `public String` **toString**`( )`
  - **Usage**
    - All implementations should provide a meaningful toString() method for debugging purposes.

12.2 Classes

12.2.1 Class **IMergeResult.Status**

Possible result status values for a merge operation.

Declaration

```
public static final class IMergeResult.Status
extends Enum
```

Fields

- public static final IMergeResult.Status FULL_MERGE
  - All data was successfully merged.
- public static final IMergeResult.Status PARTIAL_MERGE
  - Some data was successfully merged.
    Some data could not be merged and was dropped.
- public static final IMergeResult.Status NO_MERGE
  - It was not possible to merge the data.
    All clone data for the file was lost.

Chapter 13

Package core.api.provider.notification

Package Contents *Page*

13.1 Interfaces

13.1.1 Interface IEvaluationResult

Return value for `INotificationEvaluationProvider#evaluateModification(IClone, List, boolean)` .

Declaration

```
public interface IEvaluationResult
```

Methods

- *getAction*
  `public IEvaluationResult.Action` **getAction**`( )`
  - **Usage**
    - What should be done with this clone? Does the user need to be notified?
  - **Returns** - `Action` which should be taken, never null.

- *getMessage*
  `public String` **getMessage**`( )`
  - **Usage**
    - Optional notification/warning message which should be displayed to the user.
      Only applies to action types `Action#NOTIFY` and `Action#WARN` .
  - **Returns** - a human readable message, NULL if no specific message should be shown.

- *getWeight*
  `public double` **getWeight**`( )`
  - **Usage**
    - The importance of this notification/warning in relation to other events.
      The default weight is `1.0`.
      Only applies to action types `Action#NOTIFY` and `Action#WARN` .
  - **Returns** - weight of this event, >= 0.

13.1.2 Interface **INotificationDelayProvider**

A notification delay provider takes `CloneNotificationEvent` s and queues them according to some internal criteria. These `CloneNotificationEvent` s are then reexamined if certain conditions are met and are either retransmitted as new `CloneNotificationEvent` s or discarded.
The `INotificationDelayProvider` interface is implemented by all notification delay providers.

A typical implementation approach would be to queue `CloneNotificationEvent` s and to reexamine each modified clone once:
- the corresponding file was closed
- a specific amount of time has elapsed since the last modification on or near the clone

This provider type typically creates an internal background thread.

Usage Example:
Providers of this type are needed to support delayed notification of the user about some potential update anomaly. I.e. the user might still continue to modify code and might also modify the other group members of the modified clone. Displaying a warning right away might lead to superfluous warnings if the programmer is already well aware about the other clone instances.

Declaration

public interface INotificationDelayProvider
implements core.api.provider.IProvider

Methods

- *enqueueNotification*
  `public void` **enqueueNotification**`( CloneNotificationEvent` **cloneNotificationEvent** `)`
  - **Usage**
    - Takes an `IEvaluationResult` as `CloneNotificationEvent` and internally queues it for a specific time or until a specific condition arises. The corresponding `CloneNotificationEvent` is then reexamined and either dispatched as new event or discarded.

      Only one `CloneNotificationEvent` per `IClone` is queued. A new `CloneNotificationEvent` will replace any existing `CloneNotificationEvent` s for the same `IClone` .

      This method returns right away. The checking, re-examination and potential re-dispatching of the event is done in a background thread or job.

The remaining aspects, especially the criteria used for delaying and re-dispatching, are unspecified and are likely to vary from implementation to implementation.

- **Parameters**
  - `cloneNotificationEvent` - the `CloneNotificationEvent` to queue, never null. The event has to be of type `CloneNotificationEvent.Type#DELAY_NOTIFY` or `CloneNotificationEvent.Type#DELAY_WARN` . All other types are not permitted.

13.1.3 Interface **INotificationEvaluationProvider**

A notification evaluation provider is used to determine whether a specific clone modification should trigger a user notification/warning or whether it should be ignored.
The `INotificationEvaluationProvider` interface is implemented by all notification evaluation providers.

Like all providers, this implementation in itself is passive. Usually the plugin which provides the implementation will also provide some harness code which listens for `CloneModificationEvent` s and delegates the evaluation of each modified clone to this provider. The `IEvaluationResult` of the provider is then used by the harness code to update the clone data accordingly.

Declaration

public interface INotificationEvaluationProvider
implements core.api.provider.IProvider

Methods

- *evaluateModification*
  `public IEvaluationResult` **evaluateModification**`( IClone` **modifiedClone**`, List` **groupMembers**`, boolean` **initialEvaluation** `)`
  - **Usage**
    - Takes an `IClone` instance which was recently modified by the user and a list of all members of its `ICloneGroup` and evaluates how the modification should be handled.

      A notification evaluation provider may internally acquire additional information from other sources, if needed. I.e. from the registered store provider.
  - **Parameters**
    - `modifiedClone` - the clone which was modified, never null. Data on the modifications made since the last notification check are attached to the clone as an `ICloneModificationHistoryExtension` object, if this is the initial evaluation of the clone. For re-evaluations the modification history is empty. The clone itself is guaranteed to be a member of a non-empty clone group.
    - `groupMembers` - a list of all members of modifiedClone's clone group, modifiedClone itself is also part of the list, may be NULL. If this is NULL, the implementation will internally acquire the clone group data from the `IStoreProvider` .
    - `initialEvaluation` - true if this is the first time this modification is evaluated. Typically this is set to true when the modification is first seen as an `CloneModificationEvent` and set to false for later re-evaluations due to (delayed) `CloneNotificationEvent` s.
  - **Returns** - the `IEvaluationResult` for this modification, never null.
  - **See Also**
    - `IEvaluationResult`

13.2 Classes

13.2.1 Class **IEvaluationResult.Action**

The type of action which should be taken as a result of an evaluation.

Declaration

public static final class IEvaluationResult.Action
extends Enum

Fields

- public static final IEvaluationResult.Action IGNORE
  - Completely ignore this modification event.
    The state of the clone and its clone group members will not be modified.

    This action is typically chosen, if the nature of the change guarantees that an evaluation of the modification of this clone and its group members would not yield any different results than before the change.
    I.e. a white space only change or a change which only affected a comment.
- public static final IEvaluationResult.Action INSYNC
  - The clone is in sync with all its clone group members. This means that they are all semantically equivalent.

    Notifications and modified states for the clone and all its clone group members should be cleared.
    The new state for all of them would be `IClone.State#DEFAULT` .
- public static final IEvaluationResult.Action INSYNC_CUSTOMISED
  - The clone is in sync with all its clone group members, if one considers all modifications made shortly after the creation of each group member to be of no consequence.

    This state therefore describes parametrised clones which have not been modified in any significant way since their initial parametrisation.

    Notifications and modified states for the clone and all its clone group members should be cleared.
    The new state for all of them would be `IClone.State#CUSTOMISED` .
- public static final IEvaluationResult.Action MODIFIED
  - The clone modification is minor but does represent a possible change in semantics which might be of interest to the user.
    At the same time the modification is not deemed important enough to warrant an action of type `Action#NOTIFY` or `Action#WARN` .

    This clone and all other members of this clone group should be set to `IClone.State#MODIFIED` . Other group members are not updated to this state if they already have a higher state set (`IClone.State#NOTIFY` or `IClone.State#WARN`).
- public static final IEvaluationResult.Action NOTIFY

  - The user should be notified about this modification. It might have introduced some update anomalies. Notifications are typically displayed in some non intrusive manner.

    Indicates that this clone's state should be set to `IClone.State#NOTIFY` and the state of all its group members to `IClone.State#MODIFIED` , unless they already have a higher state set.

- public static final IEvaluationResult.Action WARN
  - The user should be warned about this modification. There is a very high likelihood that it has introduced some update anomalies.
    Warnings are typically displayed in a more prominent manner. They might be displayed in the same way as java warnings or errors.
    This action type should be used very sparingly.

    Indicates that this clone's state should be set to `IClone.State#WARN` and the state of all its group members to `IClone.State#MODIFIED` , unless they already have a higher state set.

- public static final IEvaluationResult.Action INSTANT_NOTIFY
  - Similar to `Action#NOTIFY` .
    But indicates to the client of the `INotificationEvaluationProvider` that this notification should be made visible to the user instantly.
    This should be used only in cases were it is obvious that user should be notified right away. The default behaviour of `Action#NOTIFY` is to allow the client to delay the notification until the user has finished modifying the clone and its surroundings. The client will then typically delegate the clone back to the `INotificationEvaluationProvider` for reevaluation once the "delay" has passed.

- public static final IEvaluationResult.Action INSTANT_WARN
  - Similar to `Action#WARN` .

- public static final IEvaluationResult.Action LEAVE_GROUP
  - The clone modification has changed the clone to an extend which makes it very likely that the clone does no longer belong to its original clone group. It should therefore be removed from the group and be treated as a stand alone instance.

Chapter 14

Package core.api.provider.reconciler

Package Contents *Page*

14.1 Interfaces

14.1.1 Interface **IDiffProvider**

A `IDiffProvider` provides character based diff services to other components.

A character based diff provides a hint at how the differences between two given text fragments may have occurred. There is no guarantee that the returned `IDiffResult` s correspond to the real modifications made.

The actual diff algorithm used is not specified and it is up to the implementation how diffs are generated.

The most prominent user of this provider is the `CPC Reconciler` module.

Declaration

```
public interface IDiffProvider
implements core.api.provider.IProvider
```

Methods

- *charDiff*
  `public List` **`charDiff`**`( String` **`oldText`**`, String` **`newText`** `)`
  - **Usage**
    - Computes a character based diff between the two given strings.
  - **Parameters**
    - `oldText` - the old text, never null.
    - `newText` - the new text, never null.
  - **Returns** - a list of differences between the two strings, never null.

14.1.2 Interface **IDiffResult**

Result wrapper object for the `IDiffProvider` .

Declaration

public interface IDiffResult

Methods

- *getLength*
  `public int` **`getLength`**`( )`
  - **Usage**
    - Retrieves the length of the added or removed text.
      Convenience method.
  - **Returns** - cached value of length of `#getText()`

- *getOffset*
  `public int` **`getOffset`**`( )`
  - **Usage**
    - Retrieves the 0-based character offset in the source text where this insertion/deletion starts.
  - **Returns** - offset in the source text where this insertion/deletion starts.

- *getText*
  `public String` **`getText`**`( )`
  - **Usage**
    - Retrieves the text which was inserted or deleted.
  - **Returns** - The text which was inserted or deleted.

- *getType*
  `public IDiffResult.Type` **`getType`**`( )`
  - **Usage**
    - Retrieves the type of this diff.
  - **Returns** - The type of this diff.

- *isDelete*
  `public boolean` **isDelete**`( )`
  - **Usage**
    - Checks whether this is a deletion.
      Convenience method.
  - **Returns** - true if this diff was a DELETE.

- *isInsert*
  `public boolean` **isInsert**`( )`
  - **Usage**
    - Checks whether this is an insertion.
      Convenience method.
  - **Returns** - true if this diff was an INSERT.

14.1.3 Interface **IReconcilerProvider**

Interface for external modification reconciliation providers.
Only one reconciler can be active at any point in time. However, a reconciler will typically allow other modules to contribute their own reconciliation sub-strategies.

Declaration

public interface IReconcilerProvider
implements core.api.provider.IProvider

Methods

- *reconcile*
  `public IReconciliationResult` **reconcile**`( ICloneFile` **cloneFile**`, List` **persistedClones**`, String` **persistedFileContent**`, String` **newFileContent**`, boolean` **notifyUser** `)`
  - **Usage**
    - Tries to reconcile an external modification of a clone file and the internal clone data.
      Takes the existing clone data and tries to identify the new positions and sizes in the new file content.
      Clones may be moved, modified or deleted. No new clones may be added by the reconciler.

      A reconciler **does not** access or modify the corresponding clone file via the file system and **does not** modify the clone data directly, i.e. via the `IStoreProvider` .
      A reconciler may not have any side effects.

      After the reconciler returns, the non-removed clone data remaining in the `IReconciliationResult` must be valid. All clones for which the new position could not be calculated must be listed in the `IReconciliationResult` 's removedClones list.

      If no reconciliation is needed, the reconciler returns a `IReconciliationResult` where all clone lists are null.
  - **Parameters**
    - `cloneFile` - the clone file which was modified, never null.
    - `persistedClones` - the currently persisted clone data for the file, never null. Clone list is **sorted by start offset**.
    - `persistedFileContent` - the currently persisted content for the file, may be NULL.

    * `newFileContent` - the new content which was produced by some external modification to the file, may be NULL.
    * `notifyUser` - true if the user should be notified about this reconciliation (or rather the fact that an external modification has taken place). The user should at least be offered two choices: try to reconcile changes or drop all clone data for file. It is up to the provider implementation to decide on a suitable way of displaying this information to the user (i.e. a simple dialog or an entire wizard).
  - **Returns** - a valid reconciliation result, never null.

14.1.4 Interface **IReconciliationResult**

Structured return value for the `IReconcilerProvider` .
The semantics of the clone lists are similar to those of the `CloneModificationEvent` .
However, they are guaranteed to be non-null at all times.

Declaration

```
public interface IReconciliationResult
```

Methods

- *getLostClones*
  `public List` **getLostClones**`( )`
  - **Usage**
    * Returns a list of clones for which the clone positions could not be reconciled.

      If `Status#FULL_RECONCILIATION` is set, this method is guaranteed to return an empty list.
      A clone which is in this list, may not be in any of the other lists.
  - **Returns** - list of lost clones, never null.

- *getModifiedClones*
  `public List` **getModifiedClones**`( )`
  - **Usage**
    * A list of clones for which the **content** was modified due to the reconciliation.
  - **Returns** - list of modified clones, never null.
  - **See Also**
    * `CloneModificationEvent.getModifiedClones()`

- *getMovedClones*
  `public List` **getMovedClones**`( )`
  - **Usage**
    * A list of clones which were moved due to the reconciliation.
      This also includes clone instances which had any other values (beside the content) modified, i.e. extension data.
  - **Returns** - list of moved clones, never null.
  - **See Also**
    * `CloneModificationEvent.getMovedClones()`

- *getRemovedClones*
  `public List` **getRemovedClones**`( )`

  - **Usage**
    - Returns the clones which were removed due to the fact that the reconciled edits removed the clone ranges from the file.

      Clones which were removed because their new positions could not be determined are **not** part of this list.

      A clone which is in this list, may not be in any of the other lists.
  - **Returns** - list of removed clones, never null.
  - **See Also**
    - `IReconciliationResult.getLostClones()`
    - `CloneModificationEvent.getRemovedClones()`

- *getStatus*
  `public IReconciliationResult.Status` **getStatus**`( )`
  - **Usage**
    - The status may only be modified by the `IReconcilerProvider` .
  - **Returns** - the final status/result of the reconciliation effort, may be NULL during reconciliation. Guaranteed to be non-null once the `IReconcilerProvider` returns.

- *isFullyReconciled*
  `public boolean` **isFullyReconciled**`( )`
  - **Usage**
    - Checks whether this result corresponds to a full reconciliation.
      Convenience method.
  - **Returns** - true if `IReconciliationResult#getStatus()` is `Status#FULL_RECONCILIATION` .

- *toString*
  `public String` **toString**`( )`
  - **Usage**
    - Each implementation should provide a meaningful toString() method.

14.2 Classes

14.2.1 Class **IDiffResult.Type**

The type of this `IDiffResult` .

Declaration

public static final class IDiffResult.Type
extends Enum

Fields

- public static final IDiffResult.Type DELETE
  - Describes a deletion.
- public static final IDiffResult.Type INSERT
  - Describes an insertion.

14.2.2 Class **IReconciliationResult.Status**

The final status/result of the reconciliation effort.

Declaration

public static final class IReconciliationResult.Status
extends Enum

Fields

- public static final IReconciliationResult.Status FULL_RECONCILIATION
  – The reconciler was able to fully reconcile all changes. No clone data was lost.
- public static final IReconciliationResult.Status PARTIAL_RECONCILIATION
  – The reconciler was able to reconcile some of the changes. Some clone data was lost.
- public static final IReconciliationResult.Status NO_RECONCILIATION
  – The reconciler was unable to reconcile the changes. All clone data was lost.

Chapter 15

Package core.api.provider.registry

Package Contents *Page*

15.1 Interfaces

15.1.1 Interface IManagableProviderRegistry

This interface lists additional methods which are required to manage a `IProviderRegistry` . As these methods are not meant for use outside of `CPCCorePlugin` but are never the less required for all potential alternative provider registry implementations, they are separately listed in this interface.

All users of the provider registry will always only see an `IProviderRegistry` interface.

Declaration

public interface IManagableProviderRegistry
implements IProviderRegistry

Methods

- *registerProvider*
  `public void` **registerProvider**`( IProviderDescriptor` **providerDescriptor** `)`
  - **Usage**

    * Registers a provider with the given priority.

      When used inside of Eclipse this method is usually not needed as the provider registry will retrieve information on the registered providers directly from the Eclipse extension point framework.
  - **Parameters**
    * `providerDescriptor` - the provider to register, never null

- *shutdown*
  `public void` **shutdown**`( )`
  - **Usage**
    * Called when the provider registry is being shut down.
      This typically only happens when the Eclipse IDE is being shutdown.

      All registered an instantiated providers need to be notified of this fact as they may need to do some cleanup work on shutdown.

- *unregisterProvider*
  `public boolean` **unregisterProvider**`( IProviderDescriptor` **providerDescriptor** `)`
  - **Usage**
    * Unregisters a provider with the registry.
      This operation has no effect if the provider was not registered.

      Only the `typeClass` and `providerClass` values of the given `IProviderDescriptor` are used. If multiple providers match these criteria, all are unregistered.
  - **Parameters**
    * `providerDescriptor` - the provider to unregister, never null
  - **Returns** - true if the provider was unregistered, false if the provider wasn't registered in the first place

15.1.2 Interface **IProviderDescriptor**

API interface for the descriptors used to handle lazy loaded `IProvider` instances in an `IProviderRegistry` implementation.

Given an `IProviderDescriptor` an instance of the corresponding provider can be obtained by calling `IProviderRegistry#lookupProvider(IProviderDescriptor)` .

Declaration

public interface IProviderDescriptor
implements Comparable

Methods

- *getName*
  `public String` **getName**`( )`
  - **Usage**
    * A human readable name for this provider.
  - **Returns** - name of this provider, never null.

- *getPriority*
  `public byte` **getPriority**`( )`
  - **Usage**
    - The priority of this provider. The higher, the more likely it is to be used.
      The priority of the default implementations is 0.
  - **Returns** - priority of this provider, may be negative.

- *getProviderClass*
  `public String` **getProviderClass**`( )`
  - **Usage**
    - The implementing class for this provider.
      This must be the fully qualified name of a class which implements the `IProviderDescriptor#getTypeClass()` interface.

      CAUTION: do not use this value to try to obtain an instance of this provider yourself. All instances need to be retrieved via the corresponding `IProviderRegistry` methods.
  - **Returns** - FQN of the implementing class for this provider, never null.

- *getTypeClass*
  `public String` **getTypeClass**`( )`
  - **Usage**
    - The API interface which this provider implements.
      This must be the fully qualified name of an `IProvider` sub-interface.
  - **Returns** - FQN of `IProvider` API interface which is implemented by this provider, never null.

- *isSingleton*
  `public boolean` **isSingleton**`( )`
  - **Usage**
    - Whether this provider is a singleton or whether a new instance is created for each lookup.

      Most providers are singletons for performance reasons.
  - **Returns** - `true` if this provider is a singleton, `false` otherwise.

- *setName*
  `public void` **setName**`( String` **name** `)`
  - **Usage**
    - Sets a human readable name for this provider.
  - **See Also**
    - `IProviderDescriptor.getName()`

- *setPriority*
  `public void` **setPriority**`( byte` **priority** `)`
  - **Usage**
    - Sets the priority of this provider.
  - **See Also**
    - `IProviderDescriptor.getPriority()`

- *setProviderClass*
  `public void` **setProviderClass**`( String` **providerClass** `)`
  - **Usage**

    - Sets the implementing class for this provider.
  - **See Also**
    - `IProviderDescriptor.getProviderClass()`

- *setSingleton*
  `public void` **setSingleton**`( boolean` **singleton** `)`
  - **Usage**
    - Specifies whether this provider is a singleton or whether a new instance is created for each lookup.
  - **See Also**
    - `IProviderDescriptor.isSingleton()`

- *setTypeClass*
  `public void` **setTypeClass**`( String` **typeClass** `)`
  - **Usage**
    - Sets the API interface which this provider implements.
  - **See Also**
    - `IProviderDescriptor.getTypeClass()`

15.1.3 Interface **IProviderRegistry**

The `IProviderRegistry` is the central point of integration for most CPC plugins/extensions. In order to allow loose coupling and easy replacement of the different CPC subsystem implementations this interface provides a central registry service which CPC subsystems use to acquire provider instances for the different CPC subsystem services.

I.e. the CPC Track plugin will use the `lookupProvider()` method to receive a CPC Store implementation which is then used for local clone storage.

Provider classes are never addressed directly, all interaction between `IProviderRegistry` based CPC subsystems should occur only via `core.api` interfaces or 3rd party API interfaces for new provider types.

A reference to the currently active `IProviderRegistry` instance can be obtained via `CPCCorePlugin#getProviderRegistry()` .

NOTE: Any class implementing this interface should also implement `IManagableProviderRegistry` .

Declaration

```
public interface IProviderRegistry
```

Methods

- *lookupProvider*
  `public IProvider` **lookupProvider**`( Class` **providerType** `)`
  - **Usage**
    - Returns the provider with the highest priority for the given type.

      Any non-null result is guaranteed to be castable to the given `providerType`.
  - **Parameters**
    - `providerType` - the type of the provider to lookup, never null

  - **Returns** - provider of `providerType` type or NULL if no such provider was found.

- *lookupProvider*
  `public IProvider` **lookupProvider**`( IProviderDescriptor` **providerDescriptor** `)`
  - **Usage**
    - Returns an instance of the provider which corresponds to the given `IProviderDescriptor` .

      The `IProviderDescriptor` needs to be compatible with the `IProviderRegistry` implementation. Custom implementations of `IProviderDescriptor` are not supported.
  - **Parameters**
    - `providerDescriptor` - a valid `IProviderDescriptor` , never null.
  - **Returns** - provider of the corresponding type or NULL if no such provider was found.
  - **See Also**
    - `IProviderRegistry.lookupProvider(Class)` (in 15.1.3, page 114)
    - `IProviderDescriptor`

- *lookupProviders*
  `public List` **lookupProviders**`( Class` **providerType** `)`
  - **Usage**
    - Returns a list of descriptors for all registered providers of the given type, ordered descending according to priority.
      I.e the first element of the list is the descriptor of the provider with the highest priority (the one which would be returned by `lookupProvider()`).

      An instance for a given `IProviderDescriptor` can be obtained by calling `#lookupProvider(IProviderDescriptor)` .
  - **Parameters**
    - `providerType` - the type of the provider to lookup, never null
  - **Returns** - list of descriptors for all providers type `providerType` or empty list if no such providers were found, never null.
  - **See Also**
    - `IProviderRegistry.lookupProvider(IProviderDescriptor)` (in 15.1.3, page 115)
    - `IProviderDescriptor`

Chapter 16

Package core.api.provider.similarity

16.1 Interfaces

16.1.1 Interface ISimilarityProvider

A similarity provider can be used to determine the percentage of similarity between two given `IClone` instances.
The `ISimilarityProvider` interface is implemented by all similarity provider implementations.

An implementation will typically offer its own extension API to allow addition, modification or removal of the strategies used to determine the similarity value.

Declaration

public interface ISimilarityProvider
implements core.api.provider.IProvider

Fields

- public static final String LANGUAGE_JAVA
  - Possible value for the `language` parameters of this interface.
    Indicates to the similarity provider that the given clone contents are **potentially** valid java source fragments.
    This is only a hint, the source fragments may have invalid syntax or may not actually be java sources.
    The similarity provider will fall back to `ISimilarityProvider#LANGUAGE_TEXT` if it can't parse the given sources.
- public static final String LANGUAGE_OTHER
  - Possible value for the `language` parameters of this interface.
    Indicates to the similarity provider that the given clone contents are **potentially** source fragments in an unknown language.

The similarity provider may try to normalise white spaces for such cases.

- public static final String LANGUAGE_TEXT
  - Possible value for the `language` parameters of this interface.
    Indicates to the similarity provider that the given clone contents are not sources in any particular programming language and that they should be handled as plain text.
- public static final String LANGUAGE_C_PLUS_PLUS
  - For future extensions.
- public static final String LANGUAGE_C
  - For future extensions.
- public static final String LANGUAGE_PERL
  - For future extensions.
- public static final String LANGUAGE_PHP
  - For future extensions.
- public static final String LANGUAGE_PYTHON
  - For future extensions.
- public static final String LANGUAGE_RUBY
  - For future extensions.
- public static final String LANGUAGE_JAVASCRIPT
  - For future extensions.

Methods

- *calculateSimilarity*
  `public int` **calculateSimilarity**`(` `String` **language**`,` `IClone` **clone1**`,` `IClone` **clone2**`,` `boolean` **transientCheck** `)`
  - **Usage**
    - Takes two clones and calculates the similarity of the two clones to each other.
      The similarity is returned as a percent value.

      Similarity is based on the contents of the given clones. The clone uuids are not taken into account. It is therefore possible to calculate the similarity between two instances of the same clone.

      A similarity provider may internally acquire a store provider to obtain additional data for the clones in question, if transientCheck is false.
      I.e. the detailed `CloneDiff` s.

      NOTE: A similarity of 100 may **only** be returned if it can be guaranteed that the two code fragments are semantically equal. Thus clients of this API can distinguish two classes of matches, =100 and <100.
  - **Parameters**
    - `language` - indication of the potential programming language of the given source fragments, never null.
    - `clone1` - the first clone to compare, never null.
    - `clone2` - the second clone to compare, never null.
    - `transientCheck` - true if the given clones might not be in sync with the store provider, in this case any implementation of this interface is forbidden to query the store provider for any additional info about the clones.

- **Returns** - similarity between the two clones, range: 0-100, 0 = no similarity, 100 = clones are semantically equal.

- *calculateSimilarity*
  `public int` **calculateSimilarity**`( String` **language**`, String` **content1**`, String` **content2** `)`
  - **Usage**
    - Simple interface for similarity calculation between two strings.
  - **Parameters**
    - `language` - indication of the potential programming language of the given source fragments, never null.
    - `content1` - content of the first clone, never null.
    - `content2` - content of the second clone, never null.
  - **Returns** - similarity between the two clones, range: 0-100, 0 = no similarity, 100 = clones are semantically equal.
  - **See Also**
    - `ISimilarityProvider.calculateSimilarity(String, IClone, IClone, boolean)`

Chapter 17

Package core.api.provider.store

Package Contents *Page*

17.1 Interfaces

17.1.1 Interface **IDebuggableStoreProvider**

An extension of the `IStoreProvider` interface which adds a couple of integrity checking and status methods which are meant to ease debugging efforts.

Implementation of this interface is optional.

Declaration

public interface IDebuggableStoreProvider
implements IStoreProvider

Methods

- *checkCacheIntegrity*
  `public boolean` **checkCacheIntegrity**`( )`
  - **Usage**
    * Executes an integrity check of the internal cache structures of the store provider implementation. This method does not throw an exception under any circumstance. Detailed information about any violated constraints is logged as level `ERROR` or `FATAL`.

      NOTE: this operation may be slow.

      DEBUG METHOD
  - **Returns** - `false` if integrity constrains have been violated, `true` otherwise.

- *checkDataIntegrity*
  `public boolean` **checkDataIntegrity**`( )`
  - **Usage**
    * Executes an integrity check of all internal data structures of the store provider implementation. This method does not throw an exception under any circumstance. Detailed information about any violated constraints is logged as level `ERROR` or `FATAL`.

      This method also executes all `checkCacheIntegrity()` checks.

      NOTE: this is a potentially VERY SLOW operation.

      An exclusive write lock is required before this method may be called.

      DEBUG METHOD
  - **Returns** - `false` if integrity constrains have been violated, `true` otherwise.
  - **Exceptions**
    * `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.

- *getCacheStats*
  `public String` **getCacheStats**`( )`
  - **Usage**
    * Returns a string which contains a number of statistics for the internal caching structures. The data contained and the formatting is up to the store provider implementation.

      DEBUG METHOD
  - **Returns** - caching statistics, never null.

17.1.2 Interface **IRemotableStoreProvider**

Extension interface for `IStoreProvider` which contains additional internal methods which must not be used by normal CPC modules.

These methods are chiefly of interest to remote store providers.

Declaration

public interface IRemotableStoreProvider
implements IStoreProvider

METHODS

- *addCloneFile*
  `public void` **addCloneFile**`( ICloneFile` **cloneFile** `)`
  - **Usage**
    - Adds a new `ICloneFile` instance to the data store.

      This method is used by remote store providers to synchronise local clone data with a remote location.

      This method is not meant to be used to create/persist new `ICloneFile` instances for files. Look at `IStoreProvider#lookupCloneFileByPath(String, String, boolean, boolean)` for that.

      An exclusive write lock is required before this method may be called.
  - **Parameters**
    - `cloneFile` - clone file instance to add, never null.
  - **Exceptions**
    - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.
  - **See Also**
    - `IStoreProvider.lookupCloneFileByPath(String, String, boolean, boolean)`

- *updateCloneFile*
  `public void` **updateCloneFile**`( ICloneFile` **cloneFile** `)`
  - **Usage**
    - Updates an existing `ICloneFile` instance in the data store.

      This method is used by remote store providers to synchronise local clone data with a remote location.

      This method is not needed for the updating of `ICloneFile` values during normal, local operation. `IStoreProvider#persistData(ICloneFile)` and `IStoreProvider#moveCloneFile(ICloneFile, String, String)` update these values.

      An exclusive write lock is required before this method may be called.
  - **Parameters**
    - `cloneFile` - clone file instance to update, the instance should already exist in local storage, never null.
  - **Exceptions**
    - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.
  - **See Also**
    - `IStoreProvider.persistData(ICloneFile)` (in 17.1.3, page 129)
    - `IStoreProvider.moveCloneFile(ICloneFile, String, String)`

17.1.3 INTERFACE IStoreProvider

A local storage provider provides persistence for arbitrary clone data objects.
The `IStoreProvider` interface specifies the API implemented by all local storage providers.

Only one local storage provider may be active at any given point in time.

All methods of this interface are thread safe.

All implementations of this interface also need to implement `IRemotableStoreProvider` .

Object Identity and Modification

All data returned by this API is detached from it's corresponding background storage. As such any returned objects can be arbitrarily modified without changing the stored versions of the objects and without affecting any other users of the API.

To persist any modifications made to an object the corresponding add/update or remove methods have to be called. Making modifications requires the ownership of an exclusive write lock (see Locking).

Locking

Read access to clone data does not require the manual acquisition of a lock. Each lookup/get method will internally acquire a shared read lock and release it at the end of the method, before returning the result.

All write accesses require exclusive locks on the entire store provider.

A **repeatable read** is only guaranteed to the lock holder **AFTER** the lock was granted and only for the duration of the lock. This means that the lock holder should **retrieve fresh versions of all clone data objects it needs after the lock was granted**.

Read locks are only held for the duration of a method call. Any **read only** users which require the data to remain unchanged during their runtime have to **acquire an exclusive write lock**. Even though they do not intend to make any modifications to the data.

The owner of the registered `IStoreProviderWriteLockHook` , if any, may hold an implicit write lock and might be concurrently modifying local cached versions of the clone data. If a read-only client needs to ensure that it receives the latest version of the clone data it has to acquire an exclusive write lock.

Once a client requests an exclusive write lock, a registered `IStoreProviderWriteLockHook` will be notified and will be given the opportunity to transfer any dirty clone data back to the store provider before the client's lock request is granted.

Clone Modification and Persistence Events

In general, the store provider is responsible for the generation of `CloneModificationEvent` s for all clone data modifications made by its clients. The events are generated once a client has relinquished its exclusive write lock.

A client may choose to handle the modification event generation itself. This fact can be communicated to the store provider by means of the `LockMode` parameter at lock acquisition time.

The store provider will furthermore generate `ClonePersistenceEvent` s whenever clone data is persisted or purged.

Design Considerations:

Care has been taken in the design of this API interface to allow maximum implementation freedom and flexibility for local storage providers. The internal storage representation is not disclosed and may vary from implementation to implementation. As the internal storage may not allow concurrent writes and may not be object oriented this enforces some API aspects which can feel somewhat cumbersome.

I.e. it is not possible to navigate through the returned object tree (`ICloneFile` and `ICloneGroup` do not hold lists of clones) and exclusive, pessimistic locking is enforced.

The normal mode of operation of the `CPC Sensor` ensures that almost all accesses to the store provider are issued from the main thread. As such lock contention is not a major issue and exclusive locks do are unlikely to introduce performance problems.

It is up to the user of the storage provider API to ensure that deadlocks are avoided.

During normal use a `StoreLockingException` will never occur. Always having to add a corresponding try/catch block may thus seem somewhat bothersome. However, it is critical that locks are always released, even in the event of other exceptions. Forcing the user of the `IStoreProvider` API to add a try/catch block around each code block which acquires an exclusive lock does reduce the likely hood of lock "leaking" quite considerably. The user will also receive direct feedback from the IDE during programming if an exclusive lock-requiring operation is used outside of such a try/catch block.

Declaration

public interface IStoreProvider
implements core.api.provider.IProvider

Methods

- *acquireWriteLock*
  `public void` **acquireWriteLock**`( IStoreProvider.LockMode` **mode** `)`
  - **Usage**
    - Acquires an exclusive write lock for the entire Store repository.

      If a user of this store provider has the intention of modifying **ANY** of the data stored by this provider it is mandatory that an exclusive write lock is obtained **BEFORE** any data is retrieved from the store provider.

      Read locks are automatically acquired and released by the lookup/get methods. Each read lock is only held for the duration of the method call. If any read only user of the storage provider needs to ensure that data is not changed over a specific period (i.e. to guarantee repeatable reads), such a read only user should also acquire an exclusive write lock.

      The owner of the registered `IStoreProviderWriteLockHook` , if any, may be holding an implicit write lock and may be modifying local cached data while no other party is holding a write lock. Such modified data will be transfered back to the store provider when a 3rd party requests a write lock. Acquiring and releasing a write lock is therefore required if a 3rd party needs to ensure that it will obtain the latest version of the data.

      This method will block until a lock can be obtained.

      Exclusive write locks are reentrant. However, the `LockMode` can't be changed during a re-entry. The lock mode used when initially acquiring the exclusive lock will remain in effect until the lock is released.

      See `LockMode` for a description of the possible modes.

      NOTE: In situations where a client needs additional locks it is advisable to ensure that either:
      - No other thread which may need exclusive `IStoreProvider` write locks is also obtaining any of these additional locks.
      - All threads will always obtain the locks in the same order.

      Otherwise deadlocks may occur.
  - **Parameters**
    - `mode` - configures specific behaviour of this lock, null is equal to LockMode.DEFAULT.
  - **Exceptions**
    - `StoreLockingException` - If maximum number of allowed re-entries is exceeded. Please also refer to the design considerations section in `IStoreProvider` .
  - **See Also**
    - `IStoreProvider.releaseWriteLock()`
    - `IStoreProvider.LockMode`

- *acquireWriteLockNonBlocking*
  `public boolean` **acquireWriteLockNonBlocking**`( IStoreProvider.LockMode` **mode**`, long` **maxWait** `)`
  - **Usage**
    - Similar to `#acquireWriteLock(LockMode)` but does not block in case the lock can not be obtained.
      If this method returns `true` an exclusive write lock was granted to this thread.
      `False` is returned if another thread already holds an exclusive write lock. In which case, this method has no effect.

      NOTE: This method does not honour any fairness conditions. The lock is re-entrant.
      NOTE: Please refer to the javadoc for `#acquireWriteLock(LockMode)` .

- **Parameters**
  - `mode` - configures specific behaviour of this lock, null is equal to LockMode.DEFAULT.
  - `maxWait` - the maximum amount of time in milliseconds that the caller is willing to wait to acquire a lock. Set to 0 to try once and fail instantly if the lock is currently held.
- **Returns** - true if the lock could be obtained, false if the lock is currently held by another thread.
- **Exceptions**
  - `StoreLockingException` - If maximum number of allowed re-entries is exceeded. Please also refer to the design considerations section in `IStoreProvider` .
  - `InterruptedException` - If the thread is interrupted while waiting for the lock.
- **See Also**
  - `IStoreProvider.acquireWriteLock(LockMode)`

- *addClone*
  `public void` **addClone**`( IClone` **clone** `)`
  - **Usage**
    - Adds a given clone to the data store. If the clone already exists, nothing is done.
      A `ICloneGroup` or `ICloneFile` referenced by the clone instance is not automatically persisted!

      If the clone contains an `ICloneModificationHistoryExtension` any `CloneDiff` objects it contains will be added to the internal modification history for this clone. The history can be retrieved via a call to `#getFullCloneObjectExtension(ICloneObject, Class)` .

      If `LockMode` does not disable events, this clone will be included in the `CloneModificationEvent#getAddedClones()` list of a `CloneModificationEvent` once the lock is released.

      An exclusive write lock is required before this method may be called.
  - **Parameters**
    - `clone` - the clone to store, never null
  - **Exceptions**
    - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.
  - **See Also**
    - `ICloneModificationHistoryExtension`
    - `IStoreProvider.acquireWriteLock(LockMode)`
    - `IStoreProvider.LockMode`

- *addCloneGroup*
  `public void` **addCloneGroup**`( ICloneGroup` **group** `)`
  - **Usage**
    - Adds the given `ICloneGroup` to the data store.

      An exclusive write lock is required before this method may be called.
  - **Parameters**
    - `group` - the group to add, never null.
  - **Exceptions**
    - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.

- *addClones*
  `public void` **addClones**`( List` **clones** `)`
  - **Usage**
    - Convenience method, see: `#addClone(IClone)` .

      An exclusive write lock is required before this method may be called.

- **Parameters**
  - `clones` - list of clones to add, may be empty, never null.
- **Exceptions**
  - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.

- *getClonesByFile*
  `public List` **getClonesByFile**`( String` **fileUuid** `)`
  - **Usage**
    - Convenience method, equals `#getClonesByFile(String, int, int)` with a `startOffset` and `endOffset` of -1.
  - **See Also**
    - `IStoreProvider.getClonesByFile(String, int, int)`

- *getClonesByFile*
  `public List` **getClonesByFile**`( String` **fileUuid**`, int` **startOffset**`, int` **endOffset** `)`
  - **Usage**
    - Retrieves all clones for a given offset range in a given file.
      Offsets start at 0.
      All clones which collide with the given offset range will be returned. Even if they are only partly included in the range.

      Automatically acquires and releases a shared read lock.
  - **Parameters**
    - `fileUuid` - UUID of the file to retrieve the clone data for, never null.
    - `startOffset` - the offset from which on all clones should be returned, any clone ending on this offset is still included. Must be >= 0.
    - `endOffset` - the offset till which all clones should be returned, any clone starting on this offset is still included. Must be >= `startOffset` or -1 for all clones till end of file.
  - **Returns** - a sorted (by start offset), distinct list of clones for the given interval. Never null.

- *getClonesByGroup*
  `public List` **getClonesByGroup**`( String` **groupUuid** `)`
  - **Usage**
    - Retrieves all clones which are part of the given clone group.

      Automatically acquires and releases a shared read lock.
  - **Parameters**
    - `groupUuid` - uuid of the group for which all clones should be returned, never null.
  - **Returns** - **unsorted** list of group members, never null.

- *getFullCloneObjectExtension*
  `public ICloneObjectExtensionLazyMultiStatefulObject` **getFullCloneObjectExtension**`( ICloneObject` **cloneObject**`, Class` **extensionClass** `)`
  - **Usage**
    - Takes an `IClone` object and an `ICloneObjectExtension` interface class and retrieves the extension for the given interface class from the clone.
      Convenience method.
      It is then passed to the `#getFullCloneObjectExtension(ICloneObject, ICloneObjectExtensionLazyMultiStatefulObject)` method.
      If no extension of that interface class type is added, NULL is returned.
  - **Parameters**
    - `cloneObject` - the clone object to which this extension is added, never null.

    * `extensionClass` - the `ICloneObjectExtension` interface class for which the currently added extension should be fully loaded, never null.
  - **Returns** - a new cloned copy of the extension with all sub-elements loaded or NULL if no extension for that interface class was present.
  - **See Also**
    * `IStoreProvider.getFullCloneObjectExtension(ICloneObject, ICloneObjectExtensionLazyMultiStatefulObject)`

- *getFullCloneObjectExtension*
  `public ICloneObjectExtensionLazyMultiStatefulObject` **getFullCloneObjectExtension**`( ICloneObject` **cloneObject**`, ICloneObjectExtensionLazyMultiStatefulObject` **extension** `)`
  - **Usage**
    * Takes an `ICloneObjectExtensionLazyMultiStatefulObject` which is only partially loaded and returns a **new cloned copy** with all sub-elements fully loaded.

      The given clone and extension instances itself are **not** modified.
  - **Parameters**
    * `cloneObject` - the clone object to which this extension is added, never null.
    * `extension` - the extension to load all sub-elements for, never null.
  - **Returns** - a new cloned copy of the extension with all sub-elements loaded, never null.

- *getPersistedCloneFileContent*
  `public String` **getPersistedCloneFileContent**`( ICloneFile` **file** `)`
  - **Usage**
    * Retrieves the persisted content of the given file. The content represents the state of the file at the time of the last call of `#persistData(ICloneFile)` .

      An exclusive write lock is required before this method may be called.
  - **Parameters**
    * `file` - the clone file to retrieve the content for, never null.
  - **Returns** - the persisted clone content or NULL if no content is available.
  - **Exceptions**
    * `StoreLockingException` - thrown if the caller does not currently hold an exclusive write lock
  - **See Also**
    * `IStoreProvider.persistData(ICloneFile)` (in 17.1.3, page 129)

- *getPersistedClonesForFile*
  `public List` **getPersistedClonesForFile**`( String` **fileUuid** `)`
  - **Usage**
    * Retrieves the persisted clone entries for the given file.
      The result represents the clone data state for the file at the time of the last call of `#persistData(ICloneFile)` .

      Use of this method will not perform any checks for external modifications of the file and will not trigger any reconciliation operations.

      This method may be slower than `#getClonesByFile(String)` .
  - **Parameters**
    * `fileUuid` - UUID of the `ICloneFile` to retrieve the clone data for, never null.
  - **Returns** - a list of persisted `IClone` instances for this file, may be empty, never null.

- *hintPurgeCache*
  `public void` **hintPurgeCache**`( ICloneFile` **file** `)`
  - **Usage**
    - All users of the store provider API are encouraged to give certain hints to the store provider implementation which can internally be used to improve performance and to reduce memory usage.

      This method should be called if it is unlikely that the clone data for a specific file will be needed again shortly. This is typically the case if the user closes the corresponding file.

      This method does not affect the persistence state of clone data in any way and does not lead to data loss. If the clone data for the given clone file is dirty, it will remain in cache.
  - **Parameters**
    - `file` - the clone file which is most likely not being accessed again in the near future

- *hintPurgeCache*
  `public void` **hintPurgeCache**`( ICloneGroup` **group** `)`
  - **Usage**
    - All users of the store provider API are encouraged to give certain hints to the store provider implementation which can internally be used to improve performance and to reduce memory usage.

      This method should be called if it is unlikely that the clone data for a specific clone group will be needed again shortly.

      This method does not affect the persistence state of clone data in any way and does not lead to data loss. If the clone data for the given clone group is dirty, it will remain in cache.
  - **Parameters**
    - `group` - the clone group which is most likely not being accessed again in the near future

- *holdingWriteLock*
  `public boolean` **holdingWriteLock**`( )`
  - **Usage**
    - Checks whether the current thread is holding the exclusive write lock for the store provider.
      This method does not block.
  - **Returns** - true if the current thread holds the exclusive write lock for the store provider, false otherwise.

- *lookupClone*
  `public IClone` **lookupClone**`( String` **cloneUuid** `)`
  - **Usage**
    - Retrieves an `IClone` object by clone uuid.
  - **Parameters**
    - `cloneUuid` - UUID of the clone to lookup, never null.
  - **Returns** - an `IClone` instance for the given UUID or NULL if no clone with this UUID was found.

- *lookupCloneFile*
  `public ICloneFile` **lookupCloneFile**`( String` **fileUuid** `)`
  - **Usage**
    - Retrieves an `ICloneFile` handle by file uuid.

      This method does not check whether the underlying file exists.
      If an `ICloneFile` with the given uuid is found in the cache or in the persistent storage, it is returned.
      Otherwise NULL is returned.

- **Parameters**
  - `fileUuid` - the file uuid to lookup, never null.
- **Returns** - an `ICloneFile` instance for the given uuid or NULL if no file with this uuid was found.

- *lookupCloneFileByPath*
  `public ICloneFile` **lookupCloneFileByPath**`( String` **project**`, String` **path**`, boolean` **createNewUuidIfNeeded**`, boolean` **followFileMove** `)`

  - **Usage**
    - Retrieves or creates an `ICloneFile` handle for the given file.

      The file itself may no longer exist. I.e. because it was recently deleted.
      In this case the method returns the old `ICloneFile` handle, if one exists in the cache or persistent storage. Otherwise it returns NULL.

      If a file was recently moved and `followFileMove` is set to `true` this method **may** also return the `ICloneFile` handle for the new location of the file. In this case the `ICloneFile` 's project and path values might not match the given `project` and `path` parameters.
      It is up to the implementation to decide whether and how long to store file move information.
  - **Parameters**
    - `project` - project the file in question belongs to, never null
    - `path` - path to the file in question, relative to project, never null
    - `createNewUuidIfNeeded` - in situations where no existing `ICloneFile` UUID can be found for the file in question, a new `ICloneFile` instance with a new UUID is generated if this value is `true`. If this value is `false`, NULL will be returned instead of generating a new UUID and `ICloneFile` instance.
      Setting this to `true` should be the default. A `false` value is useful if you only want to check whether a given file location is of interest to CPC.
    - `followFileMove` - whether the store provider should internally check if the file was moved (`true`) to another location and return the `ICloneFile` instance for the new file location if this is the case or whether NULL should be returned (`false`).
      An `IStoreProvider` is not required to support this feature. If tracking of moved files is not supported, this parameter is silently ignored.
  - **Returns** - a CloneFile instance for the given path or NULL on error (i.e. file not found/not readable)

- *lookupCloneGroup*
  `public ICloneGroup` **lookupCloneGroup**`( String` **groupUuid** `)`

  - **Usage**
    - Retrieves an `ICloneGroup` object by clone group uuid.
  - **Parameters**
    - `groupUuid` - UUID of the clone group to lookup, never null.
  - **Returns** - an `ICloneGroup` instance for the given UUID or NULL if no clone group with this UUID was found.

- *moveCloneFile*
  `public void` **moveCloneFile**`( ICloneFile` **cloneFile**`, String` **project**`, String` **path** `)`

  - **Usage**
    - Moves the given clone file to the given project and path.

      This method is used to indicate that:
      · the file was renamed or moved
      · a folder/package which contains the file was renamed or moved
      · the project which contains the file was renamed

A clone file's `uuid` is not affected by a move.
The given `ICloneFile` instance itself is not updated.

This change is instantly persisted. A call to `#persistData(ICloneFile)` is not needed.

An exclusive write lock is required before this method may be called.

- **Parameters**
  - `cloneFile` - the clone file which should be moved, never null.
  - `project` - the new project name for the file, never null.
  - `path` - the new project relative path for the file, never null.
- **Exceptions**
  - `StoreLockingException` - thrown if the caller does not currently hold an exclusive write lock

---

- *persistData*
  `public void` **persistData**`( ICloneFile` **file** `)`
  - **Usage**
    - Stores the current clone data for the given file in persistent storage.
      This should be called when a user saves a modified source file.

      This method will update the `ICloneFile#getModificationDate()` and `ICloneFile#getSize()` values.
      The given `ICloneFile` instance itself is not updated.
      If the latest version is required, it should be obtained by calling `#lookupCloneFile(String)` .

      NOTE: this method may depend on internal caching data which may only be available if all clones for the given file were loaded once via `getClonesByFile()` before any modifications on the clone data were made.

      An exclusive write lock is required before this method may be called.
  - **Parameters**
    - `file` - the file in question, never null
  - **Exceptions**
    - `StoreLockingException` - thrown if the caller does not currently hold an exclusive write lock

---

- *purgeCache*
  `public void` **purgeCache**`( )`
  - **Usage**
    - Called to indicate that the store provider implementation should purge all it's internal caches.
      This method may be called if another CPC part detects a low memory condition.

      This method does not affect the persistence state of clone data in any way and does not lead to data loss.
      If the clone data for the given clone group is dirty, it will remain in cache.

      Store provider implementations which do not allocate potentially large in-memory caches, do not need to do anything in this method.

      All store provider implementations are encouraged to keep track of available memory on their own and to reduce their memory consumption if the available system resources reach critical levels.

- *purgeData*
  `public void` **purgeData**`( )`
  - **Usage**

    * Called to indicate that the store provider implementation should delete **ALL** data.
      The result of a call to this method is equivalent to a freshly installed store provider. No data of prior sessions may remain in storage.

      This method is typically called in unit tests to ensure a defined starting state.

      If `LockMode` does not disable events, a single `CloneModificationEvent` with `CloneModificationEvent#isFullModification()` set to true and `CloneModificationEvent#getCloneFile()` set to NULL will be generated for this action once the lock is released.

      An exclusive write lock is required before this method may be called.
  - **Exceptions**
    * `StoreLockingException` - thrown if thread is now the owner of the current exclusive write lock
  - **See Also**
    * `IStoreProvider.LockMode`

- *purgeData*
  `public void` **purgeData**`( ICloneFile` **cloneFile**`, boolean` **removeCloneFile** `)`

  - **Usage**
    * Permanently removes all clones which are associated with the given file from persistent storage and from all internal cache structures. This action is irreversible.

      This modification is automatically persisted. No call to `#persistData(ICloneFile)` is required.

      If `LockMode` does not disable events, a single `CloneModificationEvent` with `CloneModificationEvent#isFullModification()` set to true for this file will be generated for this action once the lock is released.

      An exclusive write lock is required before this method may be called.
  - **Parameters**
    * `cloneFile` - the file in question, never null
    * `removeCloneFile` - if true the clone file handle will also be purged from storage. In that case a future `#lookupCloneFileByPath(String, String, boolean, boolean)` for the project and path of this file would return a new `ICloneFile` with a new unique uuid and a call to `#lookupCloneFile(String)` with the uuid of this clone file would fail. If this is false, the clone file entry will be retained and only the clone data for the file will be purged.
  - **Exceptions**
    * `StoreLockingException` - thrown if thread is now the owner of the current exclusive write lock
  - **See Also**
    * `IStoreProvider.LockMode`

- *releaseWriteLock*
  `public void` **releaseWriteLock**`( )`

  - **Usage**
    * Releases the exclusive write lock.

      Unless the generation of `CloneModificationEvent` s was disabled at lock acquisition time by the given `LockMode` , a call to this method may generate one or more `CloneModificationEvent` s.
      The **exclusive lock is relinquished first**, then all interested parties are notified about the modification and then the control is returned to the caller.

      This method may only be called by the current holder of the write lock, otherwise an error is logged and the call is ignored.

- **See Also**
  - `IStoreProvider.acquireWriteLock(LockMode)`
  - `IStoreProvider.LockMode`

- *removeClone*
  `public void` **removeClone**`( IClone` **clone** `)`
  - **Usage**
    - Removes the given clone from the data store. If it does not exist, the call is ignored.

      If `LockMode` does not disable events, this clone will be included in the `CloneModificationEvent#getRemovedClones()` list of a `CloneModificationEvent` once the lock is released.

      An exclusive write lock is required before this method may be called.
  - **Parameters**
    - `clone` - the clone to remove, never null
  - **Exceptions**
    - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.
  - **See Also**
    - `IStoreProvider.LockMode`

- *removeCloneGroup*
  `public void` **removeCloneGroup**`( ICloneGroup` **group** `)`
  - **Usage**
    - TODO:/FIXME: clarify semantics of this operation. Does it delete the clone members of this group? Does it reset them to no-group?

      An exclusive write lock is required before this method may be called.
  - **Parameters**
    - `group` -
  - **Exceptions**
    - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.

- *removeClones*
  `public void` **removeClones**`( List` **clones** `)`
  - **Usage**
    - Convenience method, see: `#removeClone(IClone)` .

      An exclusive write lock is required before this method may be called.
  - **Exceptions**
    - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.

- *revertData*
  `public void` **revertData**`( ICloneFile` **file** `)`
  - **Usage**
    - Reverts the clone data for the given file to the latest version from persistent storage.
      This is called if a user closes a modified source file without saving it or if he reverts to the saved version.

      If `LockMode` does not disable events, a single `CloneModificationEvent` with `CloneModificationEvent#isFullModification()` set to true for this file will be generated for this action once the lock is released.

      An exclusive write lock is required before this method may be called.

- **Parameters**
  - `file` - the file in question, never null
- **Exceptions**
  - `StoreLockingException` - thrown if the caller does not currently hold an exclusive write lock
- **See Also**
  - `IStoreProvider.LockMode`

- *setWriteLockHook*
  `public void` **setWriteLockHook**`( IStoreProviderWriteLockHook` **hook** `)`
  - **Usage**
    - Registers a special write lock hook which will receive a callback whenever a user of this provider requests an exclusive write lock.
      The registered hook will be called on every call to `#acquireWriteLock(LockMode)` directly **AFTER** the write lock has been granted but before the control is returned to the caller.
      Registration of a hook is optional.

      IMPORTANT: there can be **only one registered write lock hook at any given time**. Calling this method multiple times will remove any write look hook registered earlier.

      NOTE: registration of a write lock hook is **reserved for the `CPC Track` module**.

      Rationale:

      The majority of all clone data modifications is made by the `CPC Track` module. A single user action, i.e. a refactoring or source reformat can trigger thousands of clone updates. There are also some user actions inside of an editor which can potentially create a large number of clone data updates.
      In theory the existing API would be enough even for these extreme cases. However, due to performance considerations it seems highly prudent to allow a certain amount of clone modification caching inside of the `CPC Track` module.
      The write lock hook is meant as a best effort, extended-write lock for the `CPC Track` module. Once it has registered its hook it can release any exclusive lock which it might be holding on the clone data and can still continue to internally update and modify its cached clone data. Should any other party wish to modify clone data, the store provider will acquire the exclusive write lock and will then delegate control to the registered hook which in turn will give the `CPC Track` module the chance to transmit all its clone modifications back to the store provider.
      The effect is similar to a situation where the `CPC Track` module keeps a permanent exclusive lock on the store provider and only temporarily relinquishes it, whenever some other code needs to access the store provider exclusively.
  - **Parameters**
    - `hook` - a write lock hook reference, never null.

- *updateClone*
  `public void` **updateClone**`( IClone` **clone**`, IStoreProvider.UpdateMode` **mode** `)`
  - **Usage**
    - Updates a given clone in the data store. Throws an exception if the clone doesn't exist.

      If the clone contains an `ICloneModificationHistoryExtension` any `CloneDiff` objects it contains will be added to the internal modification history for this clone. The history can be retrieved via a call to `#getFullCloneObjectExtension(ICloneObject, Class)` .

      If `LockMode` does not disable events, this clone will be included in the `CloneModificationEvent#getMovedClones()` and/or `CloneModificationEvent#getModifiedClones()` list of a `CloneModificationEvent` once the lock is released. Depending on the given `UpdateMode` .

An exclusive write lock is required before this method may be called.

- **Parameters**
  - `clone` - the clone to update, never null
  - `mode` - specifies how this clone was modified, see: `UpdateMode` , never null.
- **Exceptions**
  - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.
  - `IllegalArgumentException` - thrown if a clone with that uuid doesn't exists in the data store
- **See Also**
  - `ICloneModificationHistoryExtension`
  - `IStoreProvider.LockMode`
  - `IStoreProvider.UpdateMode`

- *updateCloneGroup*
  `public void` **updateCloneGroup**`( ICloneGroup` **group** `)`
  - **Usage**
    - Updates the given group in the data store.

      An exclusive write lock is required before this method may be called.
  - **Parameters**
    - `group` - the group to update, never null.
  - **Exceptions**
    - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.

- *updateClones*
  `public void` **updateClones**`( List` **clones**`, IStoreProvider.UpdateMode` **mode** `)`
  - **Usage**
    - Convenience method, see: `#updateClone(IClone, UpdateMode)` .

      An exclusive write lock is required before this method may be called.
  - **Exceptions**
    - `StoreLockingException` - thrown if the current thread does not hold an exclusive write lock.
    - `IllegalArgumentException` - thrown if a clone with that uuid doesn't exists in the data store

17.1.4 Interface **IStoreProviderWriteLockHook**

A special interface for exclusive write lock hook callbacks registered via the `IStoreProvider#setWriteLockHook(IStoreProviderWriteLockHook)` method.

There should be no reason why any plugin besides the `CPC Track` module should implement this interface.

Declaration

```
public interface IStoreProviderWriteLockHook
```

Methods

- *aboutToGrantWriteLock*
  `public void` **aboutToGrantWriteLock**`( )`
  - **Usage**
    - The callback method which will be called once any party tries to acquire an exclusive write lock on the `IStoreProvider` . Once this method is called the write lock will already have been granted, but the control will not yet have been transfered back to the requester of the lock.
      An implementation should use a callback to this method as a chance to write back any internally cached, potentially dirty data to the `IStoreProvider` .
  - **See Also**
    - `IStoreProvider.setWriteLockHook(IStoreProviderWriteLockHook)` (in 17.1.3, page 132)

17.2 Classes

17.2.1 Class **IStoreProvider.LockMode**

Determines the behaviour of the `#acquireWriteLock(LockMode)` exclusive write lock.

Declaration

public static final class IStoreProvider.LockMode
extends Enum

Fields

- public static final IStoreProvider.LockMode DEFAULT
  - The default locking behaviour.
    A registered `IStoreProviderWriteLockHook` will be notified and will be given the opportunity to transfer any dirty clone data back to the store provider before this lock request is granted.
    `CloneModificationEvent` s for all modified clones will be generated when the lock is released.
- public static final IStoreProvider.LockMode NO_MODIFICATION_EVENT
  - A registered `IStoreProviderWriteLockHook` will be notified and will be given the opportunity to transfer any dirty clone data back to the store provider before this lock request is granted.
    No `CloneModificationEvent` s will be generated. It is up to the caller to ensure that such events are generated and dispatched as required.
- public static final IStoreProvider.LockMode NO_WRITE_LOCK_HOOK_NOTIFY
  - `CloneModificationEvent` s for all modified clones will be generated when the lock is released.
    Does not notify a registered `IStoreProviderWriteLockHook` about the lock request.

    IMPORTANT: This mode may **only** be used by the module which owns the currently registered write lock hook (in practice this means that it should not be used outside of `CPC Track`).

    Rationale:

    This mode allows the owner of the write lock hook which was registered via `#setWriteLockHook(IStoreProviderWriteLockHook)` to acquire a lock without having to filter out its own lock requests in the write lock hook.

- public static final IStoreProvider.LockMode NO_WRITE_LOCK_HOOK_NOTIFY_NO_MODIFICATION_EVENT
  - Combination of `LockMode#NO_WRITE_LOCK_HOOK_NOTIFY` and `LockMode#NO_MODIFICATION_EVENT` .
    Neither are `CloneModificationEvent` s generated nor is any registered `IStoreProviderWriteLockHook` notified about this request.

17.2.2 Class **IStoreProvider.UpdateMode**

Used to provide `#updateClone(IClone, UpdateMode)` with information on the type of modification done to the given clone.
The updateClone method may be called multiple times for the same clone, in that case the given update modes are accumulated.
A call with `UpdateMode#MOVED` and a call with `UpdateMode#MODIFIED` together are equivalent to a single call with `UpdateMode#MOVED_MODIFIED` .

Declaration

public static final class IStoreProvider.UpdateMode
extends Enum

Fields

- public static final IStoreProvider.UpdateMode MOVED
  - The clone was only moved or some other data was modified, i.e. an extension.
    Its length and contents remain unchanged.
- public static final IStoreProvider.UpdateMode MODIFIED
  - The clone content (and potentially length) was modified.
    Its offset remains unchanged.
- public static final IStoreProvider.UpdateMode MOVED_MODIFIED
  - Combination of `UpdateMode#MOVED` and `UpdateMode#MODIFIED` .
    The clones position and content was changed.

Chapter 18

Package core.api.provider.track

Package Contents *Page*

18.1 Interfaces

18.1.1 INTERFACE IFuzzyPositionToCloneMatchingProvider

API specification for fuzzy position to clone matching providers. Such a provider is used by the `CPC Track` module to check whether any existing clone matches a given position an length.

Implementations of this interface are used to find an existing clone during a copy/cut operation.

DECLARATION

```
public interface IFuzzyPositionToCloneMatchingProvider
implements core.api.provider.IProvider
```

METHODS

- *findClone*
  `public IClone` **findClone**`( ICloneFile` **cloneFile**`, List` **clones**`, int` **offset**`, int` **length** `)`
  - **Usage**
    - Checks the specified area in the given file for existing clones and returns an existing clone if it matches the specified area relatively well.

      A typical difference in area which might not be relevant to the clone itself are leading and trailing whitespaces.

- **Parameters**
  - `cloneFile` - the `ICloneFile` in question, never null.
  - `clones` - a list of all clones within the file, never null.
  - `offset` - the start offset of the area in question, always >=0.
  - `length` - the length of the area in question, always >=0.
- **Returns** - an `IClone` instance which matches (fuzzy) the given area or NULL if no such clone was found.

18.1.2 Interface **IPositionUpdateStrategyProvider**

An `IPositionUpdateStrategyProvider` specifies how clone entries should be affected by modifications to a file. I.e. whether to consider text typed next to a clone to be part of the clone or not.

The `CPC Track` module uses the main `IPositionUpdateStrategyProvider` for all clone position modifications.

Declaration

public interface IPositionUpdateStrategyProvider
implements core.api.provider.IProvider

Methods

- *extractCloneData*
  `public void` **extractCloneData**`( Position [] `**positions**`, List `**movedClones**`, List `**modifiedClones**`, List `**removedClones**`, IDocument `**document**` )`
  - **Usage**
    - Takes an array of `CPCPosition` s and extracts any clone data modifications from it.
      Detailed descriptions of each modification are added to each clone's
      `ICloneModificationHistoryExtension` .
      Clones may be part of the `movedClones` and `modifiedClones` lists at the same time.
      The `removedClones` list must always be disjunct from the `movedClones` and `modifiedClones` lists.
  - **Parameters**
    - `positions` - an array of `CPCPosition` s to extract data from, never null.
    - `movedClones` - an empty result list in which moved clones should be stored, never null.
    - `modifiedClones` - an empty result list in which modified clones should be stored, never null.
    - `removedClones` - an empty result list in which removed clones should be stored, never null.
    - `document` - optional parameter, if present any removed clone will also have its position removed from the document, may be NULL.

- *updatePositions*
  `public boolean` **updatePositions**`( DocumentEvent `**event**`, Position [] `**positions**` )`
  - **Usage**
    - Updates the given positions in-place according to the given event.
      The `IClone` elements inside of any affected `CPCPosition` object are **not** updated.

      NOTE: The signature of this interface was dictated by performance considerations. To still allow reuse in other contexts an implementation must **not** make use of any other methods of the encapsulated document object than `IDocument#get()` and `IDocument#get(int, int)` .
      A caller may provide a custom `IDocument` implementation which supports only those two methods.

      NOTE: An implementation is **not** allowed to access an `IStoreProvider` . See comments in `CPCPositionUpdater` for more details.

- **Parameters**
  - `event` - the event to process, never null. Make sure you understand the limitations of the underlying `IDocument` object if you implement this interface.
  - `positions` - an array with positions which should be updated, never null. Positions are updated in place. All positions are guaranteed to be `CPCPosition` objects.
- **Returns** - true if at least one position was modified, false otherwise.
- **See Also**
  - `CPCPosition`

Chapter 19

Package core.api.provider.xml

Package Contents *Page*

19.1 Interfaces

19.1.1 Interface **IMappingProvider**

A mapping provider implements means of mapping `IStatefulObject` data to and from a string representation. It provides "serialisation" functionality for CPC data objects.

The format of the string representation is not specified and is entirely up to the implementation.
However, each mapping provider should add some kind of special "magic" to the header which allows itself to quickly decide whether a given string is likely to be in a mapping format which it supports.

NOTE: When mapping an external string representation to CPC objects it is recommended to use the `IMappingRegistry` instead of directly accessing the default `IMappingProvider` . This ensures that "legacy" mappings can still be read, even if the default mapping provider is changed.

Declaration

public interface IMappingProvider
implements core.api.provider.IProvider

Methods

- *extractCloneObjectUuidFromString*
  `public String` **extractCloneObjectUuidFromString**`( String` **data** `)`
  - **Usage**
    * Takes a string representation which matches the ones generated by `IMappingProvider#mapToString(MappingStore, boolean)` and extracts the `ICloneObject#getUuid()` of the main object, if it exists and is of type `ICloneObject` .
      The main object is the special object which was designated as parent object on creation of the string representation or the object, if there is exactly one object in the mapping.
      Typically this is an `ICloneFile` file UUID.

      Convenience method which is provided for performance reasons in situations where only the UUID of the main entry is of interest and where parsing of the entire file is therefore not needed.
      It is up to the implementation to decide whether to implement this method separately or whether calls to this method should simply be mapped to `IMappingProvider#mapFromString(String)` .
      This API makes no guarantee that using this method does provide any performance gain. Use of this method should be considered as an optimisation hint for the implementation.

      An implementation is not required to do a full validity check of the given data structure. Thus there may be corrupted data structures for which a call to this method succeeds but a call to `IMappingProvider#mapFromString(String)` fails.
  - **Parameters**
    * `data` - a valid string representation which encodes `IStatefulObject` data, never null.
  - **Returns** - the `ICloneObject#getUuid()` of the main object of this mapping structure or NULL if the given mapping structure contains multiple elements and no element was designated as parent element or if the main element is not of type `ICloneObject` .
  - **Exceptions**
    * `MappingException` - if the given mapping structure is not valid

- *isSupportedMappingFormat*
  `public boolean` **isSupportedMappingFormat**`( String` **data** `)`
  - **Usage**
    * Does a quick check to see whether the given cpc data mapping is in a format which this mapping provider can understand.
      Typically this method will only check some magic part of the data header and will not check the entire mapping document for integrity violations.

      It is up to the implementation how to handle this method call.
      However, this is a potentially very frequently called method and any implementation should try to optimise the performance of this method as far as possible.
  - **Parameters**
    * `data` - the cpc data mapping to check, never null.
  - **Returns** - `true` if this `IMappingProvider` is compatible with the given mapping format, `false` otherwise.
    A return value of `true` does not automatically indicate that the given mapping data is well formed and valid.

- *mapFromString*
  `public MappingStore` **mapFromString**`( String` **data** `)`
  - **Usage**
    * Takes a string representation which matches the ones generated by `IMappingProvider#mapToString(MappingStore, boolean)` and builds a `MappingStore` containing the `IStatefulObject` which the given mapping represents.
  - **Parameters**
    * `data` - a valid string representation which encodes `IStatefulObject` data, never null.

  - **Returns** - a `MappingStore` with `IStatefulObject` s corresponding to the given mapping data, never null.
  - **Exceptions**
    - `MappingException` - if the given string representation is not valid

- *mapToString*
  `public String` **mapToString**`( MappingStore` **mappingStore**`, boolean` **addHeaders** `)`
  - **Usage**
    - Takes a list of `IStatefulObject` s and maps them into a string representation.
      The `MappingStore` and the contained `IStatefulObject` s may not be modified in any way by this method.
  - **Parameters**
    - `mappingStore` - a `MappingStore` which represents the `IStatefulObject` s which should be mapped to a string representation, never null.
    - `addHeaders` - whether a full set of headers should be added (`true`) or not (`false`). Some string representations may have special nesting semantics. This parameter can be used to indicate whether the result is intended to be nested within another string representation of the same type or whether it is meant to be used standalone.
      I.e. for a XML mapping, this might turn the xml preamble on or off.
  - **Returns** - string representation of the given stateful objects, never null.
  - **Exceptions**
    - `MappingException` - if the data can not be mapped

19.1.2 Interface IMappingRegistry

A registry which allows easy access to an `IMappingProvider` which supports a given cpc data mapping.
All `IMappingProvider` s which are registered with the `IProviderRegistry` are automatically known to this registry. Individual provider priorities are taken into account if multiple providers claim to support a given cpc data mapping.

For convenience reasons the mapping registry also implements part of the `IMappingProvider` interface.
However, the `IMappingProvider` is not explicitly extended here, as many of our implementations have different post conditions.

Declaration

public interface IMappingRegistry
implements core.api.provider.IProvider

Methods

- *extractCloneObjectUuidFromString*
  `public String` **extractCloneObjectUuidFromString**`( String` **data** `)`
  - **Usage**
    - Extracts the `ICloneObject#getUuid()` from the main object in the given cpc data string mapping by using the mapping provider with the highest priority which claims to support the given cpc data type.
      Convenience method.
      First does a `#lookupMappingProviderForDataFormat(String)` followed by a `IMappingProvider#extractCloneObjectUuidFromString(String)` .
      If the initial lookup fails, NULL is returned.
  - **Parameters**
    - `data` - the cpc data mapping to extract the main entries UUID from, never null.

- **Returns** - the UUID of the main `ICloneObject` entry in the given mapping or NULL. See `IMappingProvider#extractCloneObjectUuidFromString(String)` .
- **Exceptions**
  - `MappingException` - passed through from `IMappingProvider#extractCloneObjectUuidFromString(String)` .
- **See Also**
  - `IMappingRegistry.lookupMappingProviderForDataFormat(String)` (in 19.1.2, page 142)
  - `IMappingProvider.extractCloneObjectUuidFromString(String)` (in 19.1.1, page 140)

- *lookupMappingProviderForDataFormat*
  `public IMappingProvider` **lookupMappingProviderForDataFormat**`( String` **data** `)`
  - **Usage**
    - Checks whether any of the registered `IMappingProvider` s supports the given cpc data mapping.
      The check is based on `IMappingProvider#isSupportedMappingFormat(String)` .
      If multiple mapping providers support the given cpc data mapping the one with the highest priority is returned.
  - **Parameters**
    - `data` - the cpc data mapping to get a mapping provider for, never null.
  - **Returns** - an `IMappingProvider` which claims to support the given cpc data mapping or NULL if no corresponding mapping provider could be found.

- *mapFromString*
  `public MappingStore` **mapFromString**`( String` **data** `)`
  - **Usage**
    - Maps the given cpc data string mapping to a `MappingStore` using the mapping provider with the highest priority which claims to support the given cpc data type.
      Convenience method.
      First does a `#lookupMappingProviderForDataFormat(String)` followed by a `IMappingProvider#mapFromString(String)` .
      If the initial lookup fails, NULL is returned.
  - **Parameters**
    - `data` - the cpc data mapping to convert, never null.
  - **Returns** - a valid `MappingStore` if an `IMappingProvider` could be found for the given cpc data mapping, NULL otherwise.
  - **Exceptions**
    - `MappingException` - passed through from `IMappingProvider#mapFromString(String)` .
  - **See Also**
    - `IMappingRegistry.lookupMappingProviderForDataFormat(String)` (in 19.1.2, page 142)
    - `IMappingProvider.mapFromString(String)` (in 19.1.1, page 140)

19.2 Classes

19.2.1 Class MappingStore

Simple input and output wrapper for `IMappingProvider` s.

Declaration

public class MappingStore
extends Object

Constructors

- *MappingStore*
  `protected` **MappingStore**`( )`
  - **Usage**
    - For use by sub classes only.

- *MappingStore*
  `public` **MappingStore**`( ICloneFile` **cloneFile**`, List` **clones** `)`
  - **Usage**
    - Creates a new `MappingStore` instance.
      Convenience method.
  - **Parameters**
    - `cloneFile` -
    - `clones` -

- *MappingStore*
  `public` **MappingStore**`( IStatefulObject` **statefulObject** `)`
  - **Usage**
    - Creates a new `MappingStore` instance.
  - **Parameters**
    - `statefulObject` - a `IStatefulObject` , never null.

- *MappingStore*
  `public` **MappingStore**`( IStatefulObject` **statefulParentObject**`, List` **statefulChildObjects** `)`
  - **Usage**
    - Creates a new `MappingStore` instance.
  - **Parameters**
    - `statefulParentObject` - the parent `IStatefulObject` which all given child objects should be located under, never null. This is typically an `ICloneFile` instance.
    - `statefulChildObjects` - a list of `IStatefulObject` s which should be located within the given parent object, may be empty, never null. This is typically a list of `IClone` instances.

- *MappingStore*
  `public` **MappingStore**`( List` **statefulObjects** `)`
  - **Usage**
    - Creates a new `MappingStore` instance.
  - **Parameters**
    - `statefulObjects` - a list of `IStatefulObject` s, may be empty, never null.

Methods

- *getCloneFile*
  `public ICloneFile` **getCloneFile**`( )`
  - **Usage**
    - Retrieves the `ICloneFile` which was stored as parent object in this mapping store. This may fail if this mapping store has no designated parent object or if it has a different type.
      Convenience method.
  - **Returns** - an `ICloneFile` instance if the parent object exists, NULL otherwise.

  - **Exceptions**
    - `MappingException` - if the parent object does not have type `ICloneFile` .
  - **See Also**
    - `MappingStore.getStatefulParentObject()`

- *getClones*
  `public List` **getClones**`( )`
  - **Usage**
    - Retrieves the child objects as a list of `IClone` instances. This may fail if this store does other types of classes.
      Convenience method.
  - **Returns** - a list of `IClone` s, may be empty, never null.
  - **Exceptions**
    - `MappingException` - if any of the `MappingStore#getStatefulChildObjects()` is not of type `IClone` .
  - **See Also**
    - `MappingStore.getStatefulChildObjects()`

- *getStatefulChildObjects*
  `public List` **getStatefulChildObjects**`( )`
  - **Usage**
    - A list of `IStatefulObject` s which should be mapped to/where mapped from a string representation.
      The list and its elements may not be modified.
  - **Returns** - a list of `IStatefulObject` s, may be empty, never null.

- *getStatefulParentObject*
  `public IStatefulObject` **getStatefulParentObject**`( )`
  - **Usage**
    - An additional parent `IStatefulObject` for the given child objects.
      This value may be NULL.
  - **Returns** - the parent `IStatefulObject` , may be NULL.

- *toString*
  `public String` **toString**`( )`

Chapter 20

Package exports.api.exports

Package Contents *Page*

20.1 Interfaces

20.1.1 Interface IExportController

Main backend controller for the `CPC Exports` module.
Can be used to execute clone data exports via registered `IExportToolAdapter` s.

An instance can be obtained via `CPCExportsPlugin#getExportController()` .

Declaration

```
public interface IExportController
```

Methods

- *createTask*
  `public IExportTask` **createTask**`( )`
  - **Usage**
    - Creates a new empty `IExportTask` object which can then be filled with the task configuration options and data.
      Once filled, the object can be passed to `IExportController#executeExport(IProgressMonitor, IExportTask)` to execute the export task.
  - **Returns** - an empty task object, never null.
- *executeExport*
  `public IGenericStatus` **executeExport**`( IProgressMonitor` **monitor**`, IExportTask` **exportTask** `)`
  - **Usage**

    * Executes the complete export process using the `IExportToolAdapter` which corresponds to the given `IExportToolAdapterDescriptor` .

      Export Process:
      · all source files in the given projects are collected
      · the `IExportToolAdapter` is called to process/export these files
  - **Parameters**
    * `monitor` - a progress monitor for progress reporting and cancellation, may be NULL.
    * `exportTask` - the export task to process, never null.
  - **Returns** - clone export statistics, never null.
  - **Exceptions**
    * `ImportExportConfigurationOptionException` - if the given configuration options are illegal.
    * `ImportExportFailureException` - if any error occurred during the export process.
    * `InterruptedException` - if the import was cancelled by the user.

- *getRegisteredExportToolAdapters*
  `public List` **getRegisteredExportToolAdapters**`( )`
  - **Usage**
    * Retrieves a list of all currently registered `IExportToolAdapter` implementations.
  - **Returns** - list of all registered `IExportToolAdapter` implementations, never null.

20.1.2 Interface **IExportTask**

This interface represents a complete description of an export task.
It includes all configuration options and data required by the `IExportController` implementation.

An `IImportExportTask#setToolAdapter(importexport.api.generic.IGenericImportExportDescriptor)` value needs to implement `IExportToolAdapterDescriptor` in order for this task to be valid.

A new instance can be obtained from `IExportController#createTask()` .

Declaration

public interface IExportTask
implements importexport.api.generic.IImportExportTask

Chapter 21

Package exports.api.exports.adapter

Package Contents *Page*

21.1 Interfaces

21.1.1 Interface IExportToolAdapter

A `CPC Exports` interface which can be used to contribute export implementations.
An implementation of this interface is likely to be an adapter/wrapper around some kind of mapping tool.

Implementations need to be registered with the `CPC Exports` extension point `exports.exportToolAdapters`.

Declaration

```
public interface IExportToolAdapter
```

Methods

- *processExport*
  `public IExportToolAdapter.Status` **processExport**`( IProgressMonitor` **monitor**`, IExportToolAdapterTask` **exportTask**`, IExportToolAdapterResult` **exportResult** `)`
  - **Usage**

    * Executes the clone data export according to the given export task description.
- **Parameters**
    * `monitor` - optional progress monitor, may be null.
    * `exportTask` - a description of the export task at hand, never null.
    * `exportResult` - an empty result object which will be filled with some statistics, never null.
- **Returns** - the status of this export, never null.
- **Exceptions**
    * `ImportExportConfigurationOptionException` -
    * `ImportExportFailureException` -
    * `InterruptedException` -

21.1.2 Interface **IExportToolAdapterDescriptor**

Interface for descriptor objects for `IExportToolAdapter` s.

Declaration

public interface IExportToolAdapterDescriptor
implements importexport.api.generic.IGenericImportExportDescriptor

21.1.3 Interface **IExportToolAdapterResult**

Export result wrapper for the `IExportToolAdapter` .

An `IExportToolAdapter` implementation is not required to fill provide all of these statistics.
By default all values are -1.

Declaration

public interface IExportToolAdapterResult

Methods

- *getExportedCloneCount*
  `public int` **getExportedCloneCount**`( )`
  - **Usage**
    * Retrieves the number of clones exported.
  - **Returns** - number of clones exported.

- *getExportedCloneFileCount*
  `public int` **getExportedCloneFileCount**`( )`
  - **Usage**
    * Retrieves the number of exported files which contained at least one clone.
  - **Returns** - number of exported files which contained at least one clone.

- *getExportedCloneGroupCount*
  `public int` **getExportedCloneGroupCount**`( )`

  - **Usage**
    - Retrieves the number of clone groups exported.
  - **Returns** - number of clone groups exported.

- *getTotalCloneCount*
  `public int` **getTotalCloneCount**`( )`
  - **Usage**
    - Retrieves the total number of clones found in the given files.
  - **Returns** - total number of clones found in the given files.

- *getTotalCloneFileCount*
  `public int` **getTotalCloneFileCount**`( )`
  - **Usage**
    - Retrieves the number of processed files which contained at least one clone.
  - **Returns** - number of processed files which contained at least one clone.

- *getTotalCloneGroupCount*
  `public int` **getTotalCloneGroupCount**`( )`
  - **Usage**
    - Retrieves the total number of clone groups found in the given files.
  - **Returns** - total number of clone groups found in the given files.

- *setExportedCloneCount*
  `public void` **setExportedCloneCount**`( int` **exportedCloneCount** `)`
  - **Usage**
    - Sets the number of clones exported.
  - **Parameters**
    - `exportedCloneCount` - number of clones exported.

- *setExportedCloneFileCount*
  `public void` **setExportedCloneFileCount**`( int` **exportedCloneFileCount** `)`
  - **Usage**
    - Sets the number of exported files which contained at least one clone.
  - **Parameters**
    - `exportedCloneFileCount` - number of exported files which contained at least one clone.

- *setExportedCloneGroupCount*
  `public void` **setExportedCloneGroupCount**`( int` **exportedCloneGroupCount** `)`
  - **Usage**
    - Sets the number of clone groups exported.
  - **Parameters**
    - `exportedCloneGroupCount` - number of clone groups exported.

- *setTotalCloneCount*
  `public void` **setTotalCloneCount**`( int` **totalCloneCount** `)`
  - **Usage**
    - Sets the total number of clones found in the given files.
  - **Parameters**
    - `totalCloneCount` - total number of clones found in the given files.

- *setTotalCloneFileCount*
  `public void` **setTotalCloneFileCount**`( int` **totalCloneFileCount** `)`
  - **Usage**
    - Sets the number of processed files which contained at least one clone.
  - **Parameters**
    - `totalCloneFileCount` - number of processed files which contained at least one clone.

- *setTotalCloneGroupCount*
  `public void` **setTotalCloneGroupCount**`( int` **totalCloneGroupCount** `)`
  - **Usage**
    - Sets the total number of clone groups found in the given files.
  - **Parameters**
    - `totalCloneGroupCount` - total number of clone groups found in the given files.

21.1.4 Interface **IExportToolAdapterTask**

Configuration data collection object for the `IExportToolAdapter` .

NOTE: the clone data itself is not part of the export task description. The export tool adapter needs to fetch the data from the provided store provider instance. The main rationale for this is that it might not always be possible to load all clone data into memory. By providing a store provider instance an export tool adapter can process the clone data file by file.

NOTE: The caller of an `IExportToolAdapter` does **not** hold a lock on the store provider.

Declaration

public interface IExportToolAdapterTask
implements importexport.api.generic.IImportExportToolAdapterTask

Methods

- *getStoreProvider*
  `public IStoreProvider` **getStoreProvider**`( )`
  - **Usage**
    - Retrieves the `IStoreProvider` instance which should be used to obtain the clone data for the files which were selected for this export.

      NOTE: The caller of an `IExportToolAdapter` does **not** hold a lock on the store provider.
  - **Returns** - valid `IStoreProvider` reference, never null.

- *setStoreProvider*
  `public void` **setStoreProvider**`( IStoreProvider` **storeProvider** `)`
  - **Usage**
    - Sets the `IStoreProvider` instance which should be used to obtain the clone data for the files which were selected for this export.
  - **Parameters**
    - `storeProvider` - valid `IStoreProvider` reference, never null.

21.2 Classes

21.2.1 Class **IExportToolAdapter.Status**

Return value for `IExportToolAdapter#processExport(IProgressMonitor, IExportToolAdapterTask, IExportToolAdapterResult)` .

Declaration

public static final class IExportToolAdapter.Status
extends Enum

Fields

- public static final IExportToolAdapter.Status FULL_EXPORT
  - The export process finished successfully.
    All clone data was exported.
- public static final IExportToolAdapter.Status PARTIAL_EXPORT
  - No error occurred but the export did not export all clone data.
    This may happen in cases where special filters are applied during export.
- public static final IExportToolAdapter.Status NO_EXPORT
  - No error occurred but the export did not export any clone data.
    This typically happens if the exported files/projects currently don't contain any cpc clone data.

Chapter 22

Package imports.api.data

Package Contents *Page*

22.1 Interfaces

22.1.1 Interface **IImportCloneObjectExtension**

Optional clone object extension which **may** be used by `IImportToolAdapter` s to provide some per-clone confidence data.

An `IImportToolAdapter` is not required to make use of this feature.

Declaration

public interface IImportCloneObjectExtension
implements core.api.data.ICloneObjectExtension

Methods

- *getConfidence*
  `public byte` **getConfidence**`( )`
  - **Usage**
    - Provides the import implementations confidence in the detection accuracy of this clone.
  - **Returns** - confidence in percent (range: 0-100) or -1 if not set.

Chapter 23

Package imports.api.imports

23.1 Interfaces

23.1.1 Interface IImportController

Main backend controller for the `CPC Imports` module.
Can be used to execute clone data imports via registered `IImportToolAdapter` s.

An instance can be obtained via `CPCImportsPlugin#getImportController()` .

Declaration

```
public interface IImportController
```

Methods

- *createTask*
  `public IImportTask` **`createTask`**`( )`
  - **Usage**
    - Creates a new empty `IImportTask` object which can then be filled with the task configuration options and data.
      Once filled, the object can be passed to `IImportController#executeImport(IProgressMonitor, IImportTask)` to execute the import task.
  - **Returns** - an empty task object, never null.

- *executeImport*
  `public IGenericStatus` **`executeImport`**`( IProgressMonitor` **`monitor`**`, IImportTask` **`importTask`** `)`
  - **Usage**

  - Executes the complete import process using the `IImportToolAdapter` which corresponds to the given `IImportToolAdapterDescriptor` .

    Import Process:
    - all source files in the given projects are collected
    - the `IImportToolAdapter` is called to process these files
    - the returned clone data is passed to the registered `IImportFilterStrategy` s
    - the resulting clone data is transmitted to the `IStoreProvider`

- **Parameters**
  - `monitor` - a progress monitor for progress reporting and cancellation, may be NULL.
  - `importTask` - the import task to process, never null.
- **Returns** - clone import statistics, never null.
- **Exceptions**
  - `ImportExportConfigurationOptionException` - if the given configuration options are illegal.
  - `ImportExportFailureException` - if any error occurred during the import process.
  - `InterruptedException` - if the import was cancelled by the user.

- *getRegisteredImportFilterStrategies*
  `public List` **getRegisteredImportFilterStrategies**`( )`
  - **Usage**
    - Retrieves a list of all currently registered `IImportFilterStrategy` implementations.
  - **Returns** - list of all registered `IImportFilterStrategy` implementations, never null.

- *getRegisteredImportToolAdapters*
  `public List` **getRegisteredImportToolAdapters**`( )`
  - **Usage**
    - Retrieves a list of all currently registered `IImportToolAdapter` implementations.
  - **Returns** - list of all registered `IImportToolAdapter` implementations, never null.

23.1.2 Interface **IImportTask**

This interface represents a complete description of an import task.
It includes all configuration options and data required by the `IImportController` implementation.

An `IImportExportTask#setToolAdapter(importexport.api.generic.IGenericImportExportDescriptor)` value needs to implement `IImportToolAdapterDescriptor` in order for this task to be valid.

A new instance can be obtained from `IImportController#createTask()` .

Declaration

public interface IImportTask **implements** importexport.api.generic.IImportExportTask

Methods

- *getImportFilterStrategies*
  `public List` **getImportFilterStrategies**`( )`
  - **Usage**

    - Retrieves a list of descriptors of all `IImportFilterStrategy` s which should be applied to this import.
      If this is null the default strategies will be applied.
      If this is an empty list, no strategies will be applied.
  - **Returns** - list of `IImportFilterStrategyDescriptor` s, may be NULL.

- *getImportFilterStrategyOptions*
  `public Map` **getImportFilterStrategyOptions**`( )`
  - **Usage**
    - Retrieves configuration options for all `IImportFilterStrategy` s.

      Only `IImportFilterStrategy` s which defined configuration options in their extension descriptor are guaranteed to be listed in the map.

      If `IImportTask#getImportFilterStrategies()` is null, this value may also be null.
  - **Returns** - a map which **may** contain configuration option maps for each `IImportFilterStrategy` , may be NULL.

- *isClearExistingClones*
  `public boolean` **isClearExistingClones**`( )`
  - **Usage**
    - Whether existing clone data should be purged before processing the import.
  - **Returns** - true if existing clone data should be deleted

---

- *setClearExistingClones*
  `public void` **setClearExistingClones**`( boolean` **clearExistingClones** `)`
  - **Usage**
    - Specifies whether existing clone data should be purged before processing the import.
  - **Parameters**
    - `clearExistingClones` - true if existing clone data should be deleted

- *setImportFilterStrategies*
  `public void` **setImportFilterStrategies**`( List` **importFilterStrategies** `)`
  - **Usage**
    - Sets a list of descriptors of all `IImportFilterStrategy` s which should be applied to this import.
  - **Parameters**
    - `importFilterStrategies` - a list of `IImportFilterStrategyDescriptor` s, may be NULL.
  - **See Also**
    - `IImportTask.getImportFilterStrategies()`

- *setImportFilterStrategyOptions*
  `public void` **setImportFilterStrategyOptions**`( Map` **importFilterStrategyOptions** `)`
  - **Usage**
    - Sets the configuration options for all `IImportFilterStrategy` s.
  - **Parameters**
    - `importFilterStrategyOptions` - a map which **may** contain configuration option maps for each `IImportFilterStrategy` , may be NULL.

Chapter 24

Package imports.api.imports.adapter

Package Contents *Page*

24.1 Interfaces

24.1.1 Interface IImportToolAdapter

A `CPC Imports` interface which can be used to contribute import implementations.
An implementation of this interface is likely to be an adapter/wrapper around an existing clone detection tool.

Implementations need to be registered with the `CPC Imports` extension point `imports.importToolAdapters`.

Declaration

```
public interface IImportToolAdapter
```

Methods

- *processImport*
  `public IImportToolAdapter.Status` **processImport**`( IProgressMonitor` **monitor**`, IImportToolAdapterTask` **importTask**`, IImportToolAdapterResult` **importResult** `)`
  - **Usage**

    * Takes an `IImportToolAdapterTask` description which contains a list of files to import clone data for and an options map. Furthermore an empty import result wrapper object if provided for the resulting clone data.
      The options map contains user selected options. Possible options are specified using the corresponding `CPC Imports` extension point.

      Files for which the import did not produce any clones should not be added to the result map.

      The import implementation **may** attach additional confidence data to each imported clone by attaching `IImportCloneObjectExtension` instances to the clone objects.

      All exceptions which are not declared in the signature need to be caught and re-thrown as an `ImportExportFailureException` by all implementations.

      An implementation may not create its own `ICloneFile` instances. All returned instances must be created via `IStoreProvider#lookupCloneFileByPath(String, String, boolean, boolean)` . An implementation may not submit any data to the `IStoreProvider` , it should have **no side effects** (aside of those caused by the `IStoreProvider` 's lookup methods).
  - **Parameters**
    * `monitor` - progress monitor for progress reporting and cancellation, may be NULL. An implementation should start a new task and continuously update the amount of work done if this is not null. The task should be marked as finished, once the implementation is about to return control to the caller. The monitor should regularly be checked for cancellation.
    * `importTask` - the import task which contains the task description, never null.
    * `importResult` - the import result object to which detected clones and their groups should be added, never null.
  - **Returns** - the status of the import process, never null.
  - **Exceptions**
    * `ImportExportConfigurationOptionException` - if the option map contains illegal data or is missing required options. The exception message should be meaningful for the end user.
    * `ImportExportFailureException` - if the import failed for some reason. The exception message should be meaningful for the end user.
    * `InterruptedException` - of the import was cancelled by the user.

24.1.2 Interface **IImportToolAdapterDescriptor**

A descriptor which represents a registered `IImportToolAdapter` .

Declaration

public interface IImportToolAdapterDescriptor
implements importexport.api.generic.IGenericImportExportDescriptor

24.1.3 Interface **IImportToolAdapterResult**

Import result wrapper for `IImportToolAdapter` s.

Declaration

public interface IImportToolAdapterResult

Methods

- *getCloneGroups*
  `public List` **getCloneGroups**`( )`
  - **Usage**
    * The storage object for the resulting clone group data of the import process.
      An empty map will automatically be created and should be filled with result data by the `IImportToolAdapter` .
  - **Returns** - initially empty list, which is to be filled with the clone groups created during the import process, never null.

- *getCloneMap*
  `public Map` **getCloneMap**`( )`
  - **Usage**
    * The storage object for the resulting clone and clone file data of the import process.
      An empty map will automatically be created and should be filled with result data by the `IImportToolAdapter` .
  - **Returns** - initially empty map, which is to be filled with the clone results obtained during the import process, never null.

24.1.4 Interface **IImportToolAdapterTask**

Configuration data collection object for the `IImportToolAdapter` .

Declaration

public interface IImportToolAdapterTask
implements importexport.api.generic.IImportExportToolAdapterTask

24.2 Classes

24.2.1 Class **IImportToolAdapter.Status**

Return value for `IImportToolAdapter#processImport(IProgressMonitor, IImportToolAdapterTask, IImportToolAdapterResult)` .

Declaration

public static final class IImportToolAdapter.Status
extends Enum

Fields

- public static final IImportToolAdapter.Status SUCCESS
  - The import process finished successfully.
    Some clone data was imported.

- public static final IImportToolAdapter.Status NO_RESULTS
  - No error occurred but the import failed to detect any clone data fit for import.

Chapter 25

Package imports.api.imports.strategy

Package Contents *Page*

25.1 Interfaces

25.1.1 Interface **IImportFilterStrategy**

Import filter strategies are applied to the output of an `IImportToolAdapter` .
These strategies will usually be used to filter out obviously irrelevant or incorrect clones returned by an import tool adapter. They might also be used to merge or modify clones if this seems prudent.

Implementations need to be registered with the `CPC Imports` extension point "imports.importFilterStrategies".

Declaration

```
public interface IImportFilterStrategy
```

Methods

- *filterImport*
  `public IImportFilterStrategy.Status` **filterImport**`( Map` **cloneResults**`, Map` **options** `)`
  - **Usage**
    - Takes the result of an import operation and filters out clones which are deemed not be be worth importing. Clones may also be modified by this operation.

      All modifications are to be done in place, in the provided clone results map.

  - **Parameters**
    - `cloneResults` - the result map of the clone import, never null.
    - `options` - a configuration options map, never null.
  - **Returns** - whether further filters should be executed or not, never null.

25.1.2 Interface **IImportFilterStrategyDescriptor**

A descriptor which represents a registered `IImportFilterStrategy` .

Declaration

public interface IImportFilterStrategyDescriptor
implements importexport.api.generic.IGenericImportExportDescriptor

25.2 Classes

25.2.1 Class **IImportFilterStrategy.Status**

Return status for the `IImportFilterStrategy#filterImport(Map, Map)` method.

Declaration

public static final class IImportFilterStrategy.Status
extends Enum

Fields

- public static final IImportFilterStrategy.Status CONTINUE
  - The filter has finished and the next registered filter should continue with the processing and filtering of the clone results.
- public static final IImportFilterStrategy.Status BREAK
  - The filter has finished and decided that no further filters should be executed. Filter processing stops at this point.

Chapter 26

Package merge.api.strategy

Package Contents *Page*

26.1 Interfaces

26.1.1 Interface **ICloneObjectExtensionMerger**

A special merge handler for `ICloneObjectExtension` s.
The merge is internally handled by the registered `ICloneObjectExtensionMergeStrategy` s.

Declaration

```
public interface ICloneObjectExtensionMerger
```

Methods

- *merge*
  `public void` **merge**`(` `IReadableMergeTask` **mergeTask**, `IMergeResult` **mergeResult**, `ICloneObject` **localCloneObject**, `ICloneObject` **remoteCloneObject**, `ICloneObject` **baseCloneObject**, `ICloneObject` **mergedCloneObject** `)`
  - **Usage**
    - Merges the `ICloneObjectExtension` data of the given local, remote and base `ICloneObject` s. The given merged clone object is updated in place.
  - **Parameters**
    - `mergeTask` - current merge task, may not be modified, never null.
    - `mergeResult` - current merge result, may not be in its final state, may not be modified, never null.
    - `localCloneObject` - former local clone object, may not be modified, never null.
    - `remoteCloneObject` - former remote clone object, may not be modified, never null.
    - `baseCloneObject` - optional base clone object, may not be modified, may be NULL.
    - `mergedCloneObject` - new merged clone object, may be modified, never null.

26.1.2 Interface **ICloneObjectExtensionMergeStrategy**

A special support API which allows `ICloneObjectExtension` s and other modules to contribute special handling code for merging of `ICloneObjectExtension` data.

Every implementation needs to provide a **no-argument constructor**.

Implementations of this interface are treated as Singletons. An instance of each strategy will be generated at startup and will then be reused whenever needed.

Declaration

```
public interface ICloneObjectExtensionMergeStrategy
```

Methods

- *merge*
  `public ICloneObjectExtensionMergeStrategy.Status` **merge**`(` `IReadableMergeTask` **mergeTask**, `IMergeResult` **mergeResult**, `ICloneObject` **localCloneObject**, `ICloneObject` **remoteCloneObject**, `ICloneObject` **baseCloneObject**, `ICloneObject` **mergedCloneObject**, `LinkedList` **pendingLocalExtensions**, `LinkedList` **pendingRemoteExtensions**, `LinkedList` **pendingBaseExtensions** `)`
  - **Usage**
    - Takes a local and remote version of an `ICloneObject` instance with extensions and an optional base version and merges the data of supported extensions.

      The result is directly written to the given merged version of the clone object.

The pending base clone object list does not need to be completely processed. The pending local and remote clone object lists should be empty once **all** strategies have been executed.
Usually the last (”fallback”) strategy will take care of all remaining, pending local and remote clone objects.

- **Parameters**
  - `mergeTask` - the current merge task for which this merging is taking place, must not be modified in any way, never null.
  - `mergeResult` - the current merge result, this may not yet be the final merge result, must not be modified in any way, never null.
  - `localCloneObject` - the former local version of the clone object, must not be modified, never null.
  - `remoteCloneObject` - the former remote version of the clone object, must not be modified, never null.
  - `baseCloneObject` - an optional base version of the clone object, must not be modified, may be NULL.
  - `mergedCloneObject` - the new, merged version of the clone object, may be modified, never null.
  - `pendingLocalExtensions` - a list of so far unhandled former local extension objects, may be empty, may be modified, never null.
  - `pendingRemoteExtensions` - a list of so far unhandled former remote extension objects, may be empty, may be modified, never null.
  - `pendingBaseExtensions` - a list of so far unhandled base extension objects, may be empty, may be modified, never null.
- **Returns** - the status of this merge operations, see: `Status`

26.1.3 Interface **IMergeContext**

A collection of progress/status information for `IMergeStrategy` s as well as some utility functions.

Declaration

```
public interface IMergeContext
```

Methods

- *getCloneObjectExtensionMerger*
  `public ICloneObjectExtensionMerger` **getCloneObjectExtensionMerger**`( )`
  - **Usage**
    - Retrieves an `ICloneObjectExtensionMerger` instance which can be used to merge the extension data of a given clone par.
  - **Returns** - a `ICloneObjectExtensionMerger` , never null.

- *getPendingBaseClones*
  `public LinkedList` **getPendingBaseClones**`( )`
  - **Usage**
    - Retrieves a list of still unhandled base clones.
      This list may be modified by an `IMergeStrategy` .
  - **Returns** - a list of still unhandled base clones, may be empty, never null.

- *getPendingLocalClones*
  `public LinkedList` **getPendingLocalClones**`( )`
  - **Usage**

    * Retrieves a list of still unhandled local clones in their pre-merge state.
      This list may be modified by an `IMergeStrategy` .
  - **Returns** - a list of still unhandled local clones in their pre-merge state, may be empty, never null.

- *getPendingRemoteClones*
  `public LinkedList` **getPendingRemoteClones**`( )`
  - **Usage**
    * Retrieves a list of still unhandled remote clones in their pre-merge state.
      This list may be modified by an `IMergeStrategy` .
  - **Returns** - a list of still unhandled remote clones in their pre-merge state, may be empty, never null.

- *isLocalOrRemoteClonePending*
  `public boolean` **isLocalOrRemoteClonePending**`( )`
  - **Usage**
    * Checks if there are still some unhandled local or remote clones left in this context.
  - **Returns** - `true` if `IMergeContext#getPendingLocalClones()` or `IMergeContext#getPendingRemoteClones()` is not empty, `false` otherwise.

26.1.4 INTERFACE **IMergeStrategy**

Interface for `IMergeProvider` merge strategies.

Every implementation needs to provide a **no-argument constructor**.

Implementations of this interface are treated as Singletons. An instance of each strategy will be generated at startup and will then be reused whenever needed.

DECLARATION

public interface IMergeStrategy

METHODS

- *merge*
  `public IMergeStrategy.Status` **merge**`( IReadableMergeTask` **mergeTask**`, IWriteableMergeResult` **mergeResult**`, IMergeContext` **mergeContext** `)`
  - **Usage**
    * Tries to merge the `pendingLocalClones` and `pendingRemoteClones` according to the data given in the `mergeTask`.

      May additionally processes `pendingBaseClones` in case of 3-way-merges.
      See also: `IReadableMergeTask#isThreeWayMerge()` .

      The pending base clones list does not need to be completely processed. The pending local and remote clone lists should be empty once **all** strategies have been executed.
      Usually the last ("fallback") strategy will mark all remaining, pending local and remote clones as lost clones.

      The `pendingLocalClones`, `pendingRemoteClones` and `pendingBaseClones` as well as some support functions like a `ICloneObjectExtensionMerger` are available via the provided `IMergeContext` .

- **Parameters**
  - `mergeTask` - read-only description of the merge task at hand, the contents must not be modified in any way, never null.
  - `mergeResult` - a writable result wrapper which is used to incrementally build up the final merge result, never null. An `IMergeStrategy` may freely modify this object. However, care must be taken that a clone is not added multiple times to the same list. All clones which are added to the result need to be removed from **both** pending clones lists.
  - `mergeContext` - a collection of progress/status information about the current merge as well as some utility functions, never null. The context can be used to get access to lists of the still pending local/remote/base clones and to a merger for clone extensions.
- **Returns** - the `IMergeResult.Status` result status for this strategy, never null.
- **Exceptions**
  - `MergeException` - to be thrown when a serious error is detected. An exception should not be thrown if it can be expected that some other strategy might be able to handle this situation. All in all an exception should only be thrown in very extreme cases.

26.1.5 Interface **IReadableMergeTask**

A special read-only interface which corresponds to the `IMergeTask` interface.

This interface is meant to underline the fact that a `MergeTask` must not be modified by an `IMergeStrategy` . It does thus **not** extend `IMergeTask` .

Declaration

```
public interface IReadableMergeTask
```

Methods

- *getBaseCloneFile*
  `public ICloneFile` **getBaseCloneFile**`( )`
  - **Usage**
    - Retrieves the base revision of the `ICloneFile` underlying this merge task.
  - **Returns** - base revision of file, may be NULL.

- *getBaseClones*
  `public List` **getBaseClones**`( )`
  - **Usage**
    - Retrieves the list of clone data for the base revision of the source file.
  - **Returns** - list of base revision clones, may be NULL.

- *getBaseSourceFileContent*
  `public String` **getBaseSourceFileContent**`( )`
  - **Usage**
    - Retrieves the content for the base revision of the source file.
  - **Returns** - base resivison source file content, may be NULL.

- *getLocalCloneFile*
  `public ICloneFile` **getLocalCloneFile**`( )`

  - **Usage**
    - Retrieves the local `ICloneFile` underlying this merge task.
  - **Returns** - the local file, never null.

- *getLocalClones*
  `public List` **getLocalClones**`( )`
  - **Usage**
    - Retrieves the list of clone data for the local revision of the source file.
  - **Returns** - list of local clones, never null.

- *getLocalSourceFileContent*
  `public String` **getLocalSourceFileContent**`( )`
  - **Usage**
    - Retrieves the content for the local revision of the source file.
  - **Returns** - local source file content, never null.

- *getMergedSourceFileContent*
  `public String` **getMergedSourceFileContent**`( )`
  - **Usage**
    - Retrieves the merged content of the source file.
  - **Returns** - merged file content, never null.

- *getRemoteCloneFile*
  `public ICloneFile` **getRemoteCloneFile**`( )`
  - **Usage**
    - Retrieves the remote `ICloneFile` underlying this merge task.
  - **Returns** - remote file, never null.

- *getRemoteClones*
  `public List` **getRemoteClones**`( )`
  - **Usage**
    - Retrieves the list of clone data for the remote revision of the source file.
  - **Returns** - list of remote clones, never null.

- *getRemoteSourceFileContent*
  `public String` **getRemoteSourceFileContent**`( )`
  - **Usage**
    - Retrieves the content for the remote revision of the source file.
  - **Returns** - remote content of source file, never null.

- *isLocalBaseInSyncHint*
  `public boolean` **isLocalBaseInSyncHint**`( )`
  - **Usage**
    - Specifies whether the local revision of the source file is guaranteed to be in sync with the base revision.
  - **Returns** - `true` if local and base revisions are in sync, `false` otherwise.

- *isThreeWayMerge*
  `public boolean` **isThreeWayMerge**`( )`
  - **Usage**
    - Returns `true` if all base revision data is available.
      Otherwise `false` is returned. `False` is also returned if this merge task is not valid.
      Convenience method.

26.1.6 Interface **IWriteableMergeResult**

Extension interface of `IMergeResult` which allows `IMergeStrategy` s to incrementally build and modify the final `IMergeResult` of a merge operation.

All `IClone` instance lists which are part of the `IMergeResultPerspective` s and the `ICloneFile` instance which can be obtained from an `IWriteableMergeResult` (via the getters defined in `IMergeResult`) may be freely modified by an `IMergeStrategy` .

NOTE: The `IMergeResult#getMergedClones()` list **must not** be modified by a strategy. It is a cached, read-only list which is generated on demand.

Declaration

public interface IWriteableMergeResult
implements core.api.provider.merge.IMergeResult

Methods

- *addClone*
  `public void` **addClone**`( IClone` **clone**`, IWriteableMergeResult.Type` **localType**`, IWriteableMergeResult.Type` **remoteType** `)`
  - **Usage**
    * Adds the given clone to the clone lists of the local and remote perspective which correspond to the given types.
      Convenience method.
- *addCloneLocal*
  `public void` **addCloneLocal**`( IClone` **clone**`, IWriteableMergeResult.Type` **localType** `)`
  - **Usage**
    * Adds the given clone to the clone lists of the local perspective which correspond to the given type.
      Convenience method.
- *addCloneRemote*
  `public void` **addCloneRemote**`( IClone` **clone**`, IWriteableMergeResult.Type` **remoteType** `)`
  - **Usage**
    * Adds the given clone to the clone lists of the remote perspective which correspond to the given type.
      Convenience method.
- *addClones*
  `public void` **addClones**`( Collection` **clones**`, IWriteableMergeResult.Type` **localType**`, IWriteableMergeResult.Type` **remoteType** `)`
  - **Usage**
    * Adds the given clones to the clone lists of the local and remote perspective which correspond to the given types.
      Convenience method.
- *addClonesLocal*
  `public void` **addClonesLocal**`( Collection` **clones**`, IWriteableMergeResult.Type` **localType** `)`
  - **Usage**

    * Adds the given clones to the clone lists of the local perspective which correspond to the given type. Convenience method.

- *addClonesRemote*
  `public void` **addClonesRemote**`( Collection` **clones**`, IWriteableMergeResult.Type` **remoteType** `)`
  - **Usage**
    * Adds the given clones to the clone lists of the remote perspective which correspond to the given type. Convenience method.

- *setStatus*
  `public void` **setStatus**`( IMergeResult.Status` **status** `)`
  - **Usage**
    * Sets the new status for this merge.
      The default status is `IMergeResult.Status#NO_MERGE` .
  - **Parameters**
    * `status` - the new status for the merge operation, never null.

26.2 Classes

26.2.1 Class **ICloneObjectExtensionMergeStrategy.Status**

Return status indicator for `ICloneObjectExtensionMergeStrategy#merge(IReadableMergeTask, IMergeResult, ICloneObject, ICloneObject, ICloneObject, ICloneObject, LinkedList, LinkedList, LinkedList)` .

Declaration

public static final class ICloneObjectExtensionMergeStrategy.Status
extends Enum

Fields

- public static final ICloneObjectExtensionMergeStrategy.Status SKIPPED
  - Indicates that the strategy did not make any modifications to the merged clone object.
- public static final ICloneObjectExtensionMergeStrategy.Status PARTIAL
  - Indicates that the strategy made some modifications to the merged clone object.
    However, not all clone object extensions could be fully merged.
- public static final ICloneObjectExtensionMergeStrategy.Status FULL
  - Indicates that the strategy successfully merged all (remaining) clone object extensions.
    The merged clone object is potentially in it's final state.

    Further strategies will still be executed in order to have a chance to reject the result.
- public static final ICloneObjectExtensionMergeStrategy.Status BREAK
  - Indicates that this event should not be passed on to any more strategies and that the merged clone object is in it's final stage.

    A strategy will typically return this value if it detected a special situation which may confuse other strategies or if it needs to make sure that no other strategy will override its decision.

26.2.2 Class **IMergeStrategy.Status**

Return status indicator for `IMergeStrategy#merge(IReadableMergeTask, IWriteableMergeResult, IMergeContext)` .

Declaration

```
public static final class IMergeStrategy.Status
extends Enum
```

Fields

- public static final IMergeStrategy.Status SKIPPED
  - Indicates that the strategy did not make any modifications to the `IMergeResult` .
- public static final IMergeStrategy.Status PARTIAL
  - Indicates that the strategy made some modifications to the `IMergeResult` .
    However, not all clone positions could be fully merged.
- public static final IMergeStrategy.Status FULL
  - Indicates that the strategy successfully merged all (remaining) clone positions.
    The `IMergeResult` is potentially in it's final state.

    Further strategies will still be executed in order to have a chance to reject the result.
- public static final IMergeStrategy.Status BREAK
  - Indicates that this event should not be passed on to any more strategies and that the `IMergeResult` is in it's final stage.

    A strategy will typically return this value if it detected a special situation which may confuse other strategies or if it needs to make sure that no other strategy will override its decision.

26.2.3 Class **IWriteableMergeResult.Type**

Specifies how a clone was affected by the merge operation.

Declaration

```
public static final class IWriteableMergeResult.Type
extends Enum
```

Fields

- public static final IWriteableMergeResult.Type ADDED
  - The clone was newly added.
- public static final IWriteableMergeResult.Type MOVED
  - The clone's position or other attributes were changed.

- public static final IWriteableMergeResult.Type MODFIED
  - The content of the clone was modified.
- public static final IWriteableMergeResult.Type MOVED_MODIFIED
  - Combination of `Type#MOVED` and `Type#MODFIED` .
- public static final IWriteableMergeResult.Type REMOVED
  - The clone was removed due to a user modification.
- public static final IWriteableMergeResult.Type LOST
  - The clone was lost during the merge because its new position could not be determined.
- public static final IWriteableMergeResult.Type UNCHANGED
  - The clone was not affected by the merge.

Chapter 27

Package notification.api.strategy

Package Contents *Page*

27.1 Interfaces

27.1.1 Interface **INotificationEvaluationStrategy**

Interface for strategies which support the `NotificationEvaluationProvider` in reaching its decision about how to handle a given clone content modification.

Declaration

```
public interface INotificationEvaluationStrategy
```

Methods

- *evaluateModification*
  `public INotificationEvaluationStrategy.Status` **evaluateModification**`( IClone` **modifiedClone**`, List` **groupMembers**`, boolean` **initialEvaluation**`, INotificationEvaluationStrategyResult` **result** `)`
  - **Usage**
    - Takes a modified clone and a list of its clone group members and evaluates whether what kind of action should be taken.
  - **Parameters**

    * `modifiedClone` - the clone which was modified, never null.
    * `groupMembers` - a list of all group members of the clone's clone group, the modified clone itself is also part of this list, size always >=2, never null. The `IClone` instance will contain a description of the latest modifications as `CloneDiff` s inside of an `ICloneModificationHistoryExtension` object. However, no `CloneDiff` s will be available during reevaluation of an event.
    * `initialEvaluation` - true if this is the first time this modification is evaluated. Typically this is set to true when the modification is first seen as an `CloneModificationEvent` and set to false for later reevaluations due to (delayed) `CloneNotificationEvent` s.
    * `result` - an initially empty wrapper of `IEvaluationResult` s. The strategy should add its own results as a new `IEvaluationResult` to this wrapper. A strategy may add multiple `IEvaluationResult` s. Never null.
  - **Returns** - the `Status` of this evaluation, never null.

27.1.2 Interface **INotificationEvaluationStrategyResult**

Result wrapper for `INotificationEvaluationStrategy` implementations.

Declaration

```
public interface INotificationEvaluationStrategyResult
```

Methods

- *add*
  `public void` **add**`( IEvaluationResult` **result** `)`
  - **Usage**
    * Adds the given `IEvaluationResult` to this result collection.
      A strategy may call this method more than once.
  - **Parameters**
    * `result` - the `IEvaluationResult` to add, never null.

27.2 Classes

27.2.1 Class **INotificationEvaluationStrategy.Status**

Return status indicator for `INotificationEvaluationStrategy#evaluateModification(IClone, List, boolean, INotificationEvaluationStrategyResult)` .

Declaration

```
public static final class INotificationEvaluationStrategy.Status
extends Enum
```

Fields

- public static final INotificationEvaluationStrategy.Status SKIPPED

  – Indicates that the strategy does not apply to the given clone and did not make any modifications.

- public static final INotificationEvaluationStrategy.Status MODIFIED
  – Indicates that the strategy made some modifications to the result object.
- public static final INotificationEvaluationStrategy.Status BREAK
  – Indicates that this event should not be passed on to any more strategies and that the clone's evaluation is in it's final stage.

  A strategy will typically return this value if it detected a special situation which may confuse other strategies or if it needs to make sure that no other strategy will override its decision.

Chapter 28

Package reconciler.api.strategy

Package Contents *Page*

28.1 Interfaces

28.1.1 Interface **IReconcilerStrategy**

Interface for `IReconcilerProvider` reconciliation strategies.

Every implementation needs to provide a **no-argument constructor**.

Implementations of this interface are treated as Singletons. An instance of each strategy will be generated at startup and will then be reused whenever needed.

Declaration

```
public interface IReconcilerStrategy
```

Methods

- *reconcile*
  `public IReconcilerStrategy.Status` **reconcile**`( ICloneFile` **cloneFile**`, List` **persistedClones**`, String` **persistedFileContent**`, String` **newFileContent**`, List` **differences**`, LinkedList` **pendingClones**`, IReconciliationResult` **result** `)`
  - **Usage**
    * This method is called for each registered reconciliation strategy in turn (in order of their priority) until one method returns `Status#BREAK` or all strategies have been called.

Only the `pendingClones` and `result` parameters may be modified.
All other parameters are to be considered **read only**.

A clone to be added to the `result` may **only** be taken from the `pendingClones` list or from within the `result` object. As only that list and the `result` object are guaranteed to contain cloned instances.
The elements of the `persistedClones` list are not cloned and may not be modified.
A strategy **must not clone any objects itself**.

A strategy should ignore any clone which it can't find. Only clones which were definitely removed should be added to the `IReconciliationResult` removed clones list.
Once all strategies have been run or one strategy returned `Status#BREAK` , all remaining clones in the `pendingClones` list, will be added to the lost clones list automatically.

– **Parameters**
  * `cloneFile` - the clone file which was modified, never null.
  * `persistedClones` - the currently persisted clone data for the file, never null. Clone list is **sorted by start offset**.
  * `persistedFileContent` - the currently persisted content for the file, may be NULL.
  * `newFileContent` - the new content which was produced by some external modification to the file, may be NULL.
  * `differences` - the differences between the persisted and the new file content, never null. Differences are sorted by start offset.
  * `pendingClones` - a list of clones which have not yet been reconciled, this is the "todo list" for each strategy. A strategy should remove any clones which it adds to the `result` from this list. Clone list is **sorted by start offset**. This is the only parameter besides `result` which may be modified.
  * `result` - the current, incrementally filled `IReconciliationResult` . This is the only parameter besides `pendingClones` which may be modified.

– **Returns** - a status flag indicating the type of modifications done by this strategy.

28.2 Classes

28.2.1 Class **IReconcilerStrategy.Status**

Return status indicator for `IReconcilerStrategy#reconcile(ICloneFile, List, String, String, List, LinkedList, IReconciliationResult)` .

Declaration

public static final class IReconcilerStrategy.Status
extends Enum

Fields

- public static final IReconcilerStrategy.Status SKIPPED
  – Indicates that the strategy did not make any modifications to the `IReconciliationResult` .
- public static final IReconcilerStrategy.Status PARTIAL
  – Indicates that the strategy made some modifications to the `IReconciliationResult` .
  However, not all clone positions could be fully reconciled.
- public static final IReconcilerStrategy.Status FULL

  - Indicates that the strategy successfully reconciled all (remaining) clone positions.
    The `IReconciliationResult` is potentially in it's final state.

    Further strategies will still be executed in order to have a chance to reject the result.

- public static final IReconcilerStrategy.Status BREAK
  - Indicates that this event should not be passed on to any more strategies and that the `IReconciliationResult` is in it's final stage.

    A strategy will typically return this value if it detected a special situation which may confuse other strategies or if it needs to make sure that no other strategy will override its decision.

Chapter 29

Package similarity.api.strategy

Package Contents *Page*

29.1 Interfaces

29.1.1 Interface **ISimilarityStrategy**

Strategy extension interface for the default `ISimilarityProvider` implementation.

Declaration

```
public interface ISimilarityStrategy
```

Methods

- *calculateSimilarity*
  `public ISimilarityStrategy.Status` **calculateSimilarity**`( IStoreProvider` **storeProvider**`, ISimilarityStrategyTask` **task** `)`
  - **Usage**
    - Applies this strategy to the given similarity evaluation task.
  - **Parameters**
    - `storeProvider` - an optional store provider reference, NULL if the client requested a transient handling of the evaluation.

    - `task` - the similarity evaluation task, never null. A strategy directly modifies the task object in order to store its results.
  - **Returns** - the status of this operation, never null.
  - **See Also**
    - `ISimilarityStrategyTask`

29.1.2 INTERFACE **ISimilarityStrategyTask**

Parameter value for `ISimilarityStrategy#calculateSimilarity(core.api.provider.store.IStoreProvider, ISimilarityStrategyTask)` .

DECLARATION

```
public interface ISimilarityStrategyTask
```

FIELDS

- public static final int PROCESSING_STATUS_FILTERED
  - Set once an `ISimilarityStrategy` has filtered out parts of the the processed contents. I.e. comments.
- public static final int PROCESSING_STATUS_NORMALISED_WHITESPACE
  - Set once an `ISimilarityStrategy` has normalised white spaces of the processed contents.
- public static final int PROCESSING_STATUS_NORMALISED_IDENTIFIERS
  - Set once an `ISimilarityStrategy` has normalised identifiers of the processed contents.

METHODS

- *addScore*
  `public void` **addScore**`( double` **score**`, double` **weight** `)`
  - **Usage**
    - This method may be called only once per strategy.
  - **Parameters**
    - `score` -
    - `weight` -

- *getAverageScore*
  `public int` **getAverageScore**`( )`
  - **Usage**
    - Retrieves the current average weighted score of all strategies which have been executed for this task. Guaranteed to never return 100 if `ISimilarityStrategyTask#isForceNonEqual()` is true.
  - **Returns** - weighted, average score as integer between 0 and 100.

- *getClone1*
  `public IClone` **getClone1**`( )`
  - **Usage**
    - Retrieves the first clone for this comparison.
      The returned clone object may not be modified in any way.

  - **Returns** - first clone, never null.

- *getClone2*
  `public IClone` **getClone2**`( )`
  - **Usage**
    - Retrieves the second clone for this comparison.
      The returned clone object may not be modified in any way.
  - **Returns** - second clone, never null.

- *getLanguage*
  `public String` **getLanguage**`( )`
  - **Usage**
    - Retrieves the language for the given clone fragments.
  - **Returns** - the language for the given clone fragments, as specified by `ISimilarityProvider` , never null.

- *getProcessedContent1*
  `public String` **getProcessedContent1**`( )`
  - **Usage**
    - Retrieves the processed/normalised content for clone1.
      This value may be updated by each strategy in turn.
  - **Returns** - the current processed content for clone1, never null.

- *getProcessedContent2*
  `public String` **getProcessedContent2**`( )`
  - **Usage**
    - Retrieves the processed/normalised content for clone2.
  - **See Also**
    - `SimilarityStrategyTask.getProcessedContent1()`

- *getProcessingStatus*
  `public int` **getProcessingStatus**`( )`
  - **Usage**
    - Retrieves the processing status of the two processed content strings.
      The value is a bit mask, comprised of `PROCESSING_STATUS_*` values.
  - **Returns** - current processing status bit mask for both content strings.
  - **See Also**
    - `ISimilarityStrategyTask.setProcessedContent1(String, int)`

- *isForceNonEqual*
  `public boolean` **isForceNonEqual**`( )`
  - **Usage**
    - Whether the final similarity between the two clones needs to be capped at 99%.
      This is required to adhere to the `ISimilarityProvider` API specification which states that a similarity of 100% must only be returned if the two code sections are guaranteed to be semantically equivalent.
      If this value is true, some strategy detected a violation of that semantic equivalence.
  - **Returns** - true if this task must never return 100, false otherwise.

- *markForceNonEqual*
  `public void` **markForceNonEqual**`( )`
  - **Usage**

    - A strategy **has to** call this method if it detects any semantic difference between the two code fragments. Once set, this flag can't be unset.
  - **See Also**
    - `ISimilarityStrategyTask.isForceNonEqual()`

- *setProcessedContent1*
  `public void` **setProcessedContent1**`( String` **processedContent1**`, int` **processingStatus** `)`
  - **Usage**
    - Updates the processed/normalised content for clone1.
      Strategies may only update this value. The value of `IClone#getContent()` may not be modified.
  - **Parameters**
    - `processedContent1` - the new content for clone1, never null.
    - `processingStatus` - bit mask which indicates the kinds of modifications made to the content, data is additive. There is no need to pass through bits set earlier. Bits can't be unset. There is one bit mask for both contents.

- *setProcessedContent2*
  `public void` **setProcessedContent2**`( String` **processedContent2**`, int` **processingStatus** `)`
  - **Usage**
    - Updates the processed/normalised content for clone2.
  - **See Also**
    - `ISimilarityStrategyTask.setProcessedContent1(String, int)`

29.2 Classes

29.2.1 Class ISimilarityStrategy.Status

Return value for `ISimilarityStrategy#calculateSimilarity(IStoreProvider, ISimilarityStrategyTask)` .

Declaration

public static final class ISimilarityStrategy.Status
extends Enum

Fields

- public static final ISimilarityStrategy.Status SKIPPED
  - The strategy was not applicable. No modifications were made.
    Processing should continue with the next strategy.
- public static final ISimilarityStrategy.Status CONTINUE
  - The strategy was applied. Some modifications were made.
    Processing should continue with the next strategy.
- public static final ISimilarityStrategy.Status BREAK
  - Processing of further similarity strategies should be aborted.